(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
10 May 2001 (10.05.2001)

PCT

(10) International Publication Number
WO 01/33369 A1

(51) International Patent Classification[7]: G06F 13/00, 13/14, 15/16, 17/30, 17/60

(21) International Application Number: PCT/US00/30068

(22) International Filing Date:
2 November 2000 (02.11.2000)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
60/163,020 2 November 1999 (02.11.1999) US

(63) Related by continuation (CON) or continuation-in-part (CIP) to earlier application:
US 60/163,020 (CON)
Filed on 2 November 1999 (02.11.1999)

(71) Applicant *(for all designated States except US)*: **COMMERCE ONE** [US/US]; 4440 Rosewood Drive, Pleasanton, CA 94588-3050 (US).

(72) Inventors; and

(75) Inventors/Applicants *(for US only)*: **MELTZER, Bart, Alan** [US/US]; Aptos, CA 95003 (US). **DAVIDSON, Andrew** [US/US]; Boulder Creek, CA (US). **VENKAT, Ramshankar** [US/US]; Milpitas, CA (US). **MULLER, Tom** [US/US]; Fremont, CA (US). **ROSENTHAL, Karen** [US/US]; Knightsen, CA (US). **SCHWARZHOFF, Kelly** [US/US]; Palo Alto, CA (US). **AHMED, Zahid** [US/US]; Redwood City, CA (US).

(74) Agent: **BEFFEL, Ernest, J.**; Haynes & Beffel LLP, P.O. Box 366, Half Moon Bay, CA 94019 (US).

(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CR, CU, CZ, DE, DK, DM, DZ, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, US, UZ, VN, YU, ZA, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian

*[Continued on next page]*

(54) Title: COMMERCE COMMUNITY SCHEMA FOR THE GLOBAL TRADING WEB

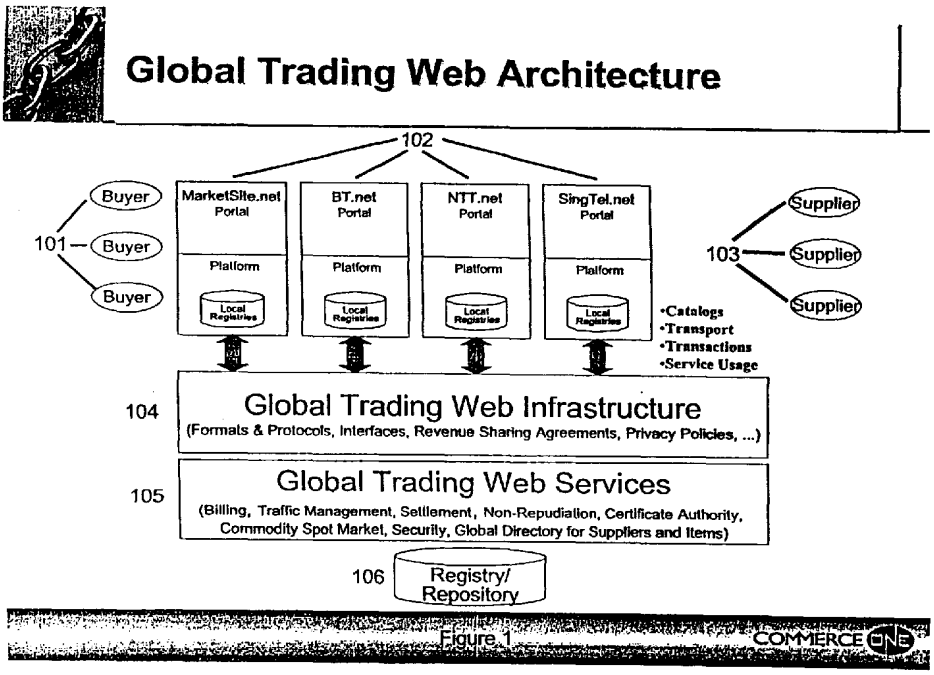


Figure 1

(57) Abstract: Machine readable documents connect business with customers (101), suppliers (103) and trading partners (102). To connect entities across marketplaces (102) to which they subscribe, there must be an interface (104) for defining documents. The present invention includes an extensible interface (104) for common definitions and vocabulary to allows marketplaces (102) to reveal their trading documents to others according to a commonly understood and marketplace independent definition.

WO 01/33369 A1

patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

**Published:**

— *With international search report.*

— *Before the expiration of the time limit for amending the claims and to be republished in the event of receipt of amendments.*

*For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.*

# COMMERCE COMMUNITY SCHEMA FOR THE GLOBAL TRADING WEB

## COPYRIGHT NOTICE



## BACKGROUND OF THE INVENTION

### Field of the Invention

The present invention relates to systems and protocols supporting transactions among diverse clients coupled to a network; and more particularly to an extensible schema for electronic data interchange among participants in a commerce community and between commerce communities.

### Description of Related Art

The Internet and other communications networks provide avenues for communication among people and computer platforms which are being used for a wide variety of transactions, including commercial transactions in which participants buy and sell goods and services. Many efforts are underway to facilitate commercial transactions on the Internet. However, with many competing standards, in order to execute a transaction, the parties to the transaction must agree in advance on the protocols to be utilized, and often require custom integration of the platform architectures to support

such transactions. Commercial processes internal to a particular node not compatible with agreed-upon standards, may require substantial rework for integration with other nodes. Furthermore, as a company commits to one standard or the other, the company becomes locked-in to a given standardized group of transacting parties, to the exclusion of others.

A good overview of the challenges met by Internet commerce development is provided in Tenenbaum et al., *"Eco System: An Internet Commerce Architecture"*, Computer, May 1997, pp. 48-55.

To open commercial transactions on the Internet, standardization of architectural frameworks is desired. Platforms developed to support such commercial frameworks include IBM Commerce Point, Microsoft Internet Commerce Framework, Netscape ONE (Open Network Environment), Oracle NCA (Network Computing Architecture), and Sun/JAVASoft JECF (JAVA Electronic Commerce Framework).

In addition to these proprietary frameworks, programming techniques, such as common distributed object model based on CORBA IIOP (Common Object Request Broker Architecture Internet ORB Protocol), are being pursued. Use of the common distributed object model is intended to simplify the migration of enterprise systems to systems which can inter-operate at the business application level for electronic commerce. However, a consumer or business using one framework is unable to execute transactions on a different framework. This limits the growth of electronic commerce systems.

Companies implementing one framework will have an application programming interface API which is different than the API's supporting other frameworks. Thus, it is very difficult for companies to access each other's business services, without requiring adoption of common business system interfaces. The development of such business system interfaces at the API level requires significant cooperation amongst the parties which is often impractical.

Accordingly, it is desirable to provide a framework which facilitates interaction amongst diverse platforms in a communication network. Such system should facilitate spontaneous commerce between trading partners without custom integration or prior agreement on industry-wide standards. Further, such systems should encourage incremental path to business automation, to eliminate much of the time, cost and risks of traditional systems integration.

Overall, it is desirable to provide an electronic commerce system that replaces the closed trading partner networks based on proprietary standards with open markets.

## SUMMARY OF THE INVENTION

The present invention is part of an infrastructure for connecting businesses with customers, suppliers and trading partners. Under the infrastructure of the present invention, companies exchange information and services using self-defining, machine-readable documents, such as XML (Extensible Markup Language) based documents, that can be easily understood amongst the partners. Documents which describe the documents to be exchanged, called business interface definitions (BIDs herein), are posted on the Internet, or otherwise communicated to members of the network. The business interface definitions tell potential trading partners the services the company offers and the documents to use when communicating with such services. Thus, a typical business interface definition allows a customer to place an order by submitting a purchase order, compliant with a document definition published in the BID of a party to receive the purchase order. A supplier is allowed to check availability by downloading an inventory status report compliant with a document definition published in the BID of a business system managing inventory data. Use of predefined, machine-readable business documents provides a more intuitive and flexible way to access enterprise applications. Consistent schema maintained by a global trading infrastructure assure reliable exchange of documents and document containers among marketplaces, whether the infrastructure participation is real or virtual.

## BRIEF DESCRIPTION OF THE FIGURES

Fig. 1 is a an overview of a GTW (Global Trading Web) architecture.

Fig. 2 is a simplified diagram of a GTW topology.

Fig. 3 is a simplified diagram of an electronic commerce network including business interface definitions BIDs according to the present invention.

Fig. 4 is a conceptual block diagram of GTW implementation options.

Fig. 5 is a high level block diagram of a registry and repository architecture.

Fig. 6 is a block diagram of entities and relationships in the commerce community schema of the present invention.

Fig. 7 is a tree structure that places in context services and applications of the present invention.

Fig. 8 is a simplified diagram of a business interface definition document according to the present invention.

Fig. 9 is a conceptual block diagram of a server for a participant node in the network of the present invention.

Fig. 10 is a flow chart illustrating the processing of a received document at a participant node according to the present invention.

Fig. 11 is a block diagram of a parser and transaction process front end for an XML based system.

Fig. 12 is a conceptual diagram of the flow of a parse function.

Fig. 13 is a simplified diagram of the resources at a server used for building a business interface definition according to the present invention.

Fig. 14 is a simplified diagram of a repository according to the present invention for use for building business interface definitions.

Fig. 15 is a flow chart illustrating the processes of building a business interface definition according to the present invention.

Fig. 16 provides a heuristic view of a repository according to the present invention.

Fig. 17 is a simplified diagram of the resources at a server providing the market maker function for the network of the present invention based on business interface definitions.

Fig. 18 is a flow chart for the market maker node processing of a received document.

Fig. 19 is a flow chart illustrating the process of registering participants at a market maker node according to the present invention.

Fig. 20 is a flow chart illustrating the process of providing service specifications at a market maker node according to the process of Fig. 15.

Fig. 21 is a diagram illustrating the sequence of operation at a participant or market maker node according to the present invention.

Fig. 22 is a conceptual diagram of the elements of a commercial network based on BIDs, according to the present invention.

## DETAILED DESCRIPTION

A detailed description of embodiments practicing the present invention is provided with respect to the figures. Other embodiments practicing the present invention will be apparent to those of skill in the art. Fig. 1 depicts a global trading Web infrastructure in support of an overall global trading Web architecture. Buyers 101 interact with market portals 102 to communicate with suppliers 103. The buyers, suppliers and other interacting through the market portals or nodes may collectively be referred to as traders. Each market portal operates from an URL and is hosted by a platform, which may be a workstation, minicomputer or mainframe running an operating system such as Windows NT, Linux, UNIX, or VM. Each platform has access to a local registry, which contains a schema representing the businesses participating in the market portal, the services they offer, interactions or transactions among businesses, the documents which direct interactions and the information items in the documents. From the perspective of a buyer or supplier, the market portals support catalogs, the transport of documents, transactions between buyers and suppliers and between marketplaces, and services in support of the transactions. The global trading Web infrastructure 104 supports the market portals that communicate with it. The infrastructure receives, stores and reveals formats and protocols. It supports interfaces between market portals that may otherwise be incompatible. It stores and assists in implementation of revenue-sharing agreements between market portals for buyers and sellers, it stores and assists in implementation of privacy policies. The communications between market portals and the infrastructure are secure, as is the platform including one or more servers that supports the infrastructure.

The market portals connected through the global trading Web infrastructure can be based on servers built on the CommerceOne TM MarketSite product line, and eCo interoperability specification compliant marketplace, or a custom marketplace solution that has been adapted for interoperability, such as Chemdex TM or any other non-MarketSite enabled marketplace. The infrastructure generally provides program business community services, a routing infrastructure for business documents and a Web portal for access to routing infrastructure and services. This infrastructure resembles the structure that may be implemented for a single MarketSite TM. It is different because the business community services provided facilitate global trading rather than trading within one community. The routing and tracking documents account for transactions

between marketplaces, rather than within a single portal. The Web portal is tailored to connecting other market Web portals around the world. For document routing, the infrastructure can be real or virtual. When it is real, the documents actually go through the infrastructure. That is, the infrastructure will act as a messaging switch, receiving and forwarding document containers and documents from one market portal to another. When it is virtual, documents are transferred in a peer-to-peer fashion, with the benefit of local registries being updated from the global registry to assure that the market portals supported are operating on current schemas and protocols. The infrastructure preferably is informed of interconnects and transfers for tracking purposes when documents are transferred peer-to-peer.

The infrastructure is useful in establishing interconnections between marketplaces. Revenue-sharing agreements may be based on infrastructure supplied forms. The billing capabilities exist to support the revenue-sharing agreements. A billable event can be logged every time an event occurs which is billable to one of the participants in the infrastructure. The infrastructure registers trading partners (also referred to as traders) and interconnections. A global namespace uniquely identifies trading partners via trading partner profiles. A privacy policy protects trading community members profiles to the extent that they assert that an interest in privacy. The infrastructure can support anonymous transactions pursuant to terms and conditions which assure payment and delivery of goods. Trading partners who are interested in being identified appear in a super directory of trading partners. Information regarding each of the trading partners is available through a browsing interface and a query mechanism. When potential trading partners find each other, either through the super directory or other means, they may wish to establish a trading relationship. The infrastructure facilitates conversations between the entities and is available to register whatever agreements they reach to support commerce between them. The infrastructure assists in reconciling mappings and other issues related to interconnections by providing businesses, market communities and context services targeted at global trading.

Operationally, the infrastructure provides activation and configuration of interconnections, maintains interconnections over time, manages relationships across interconnections and tracks activity for billing and confirmation purposes. These operational capabilities depend on the consistency provided by the infrastructure in formats, protocols, policies (e.g., privacy), processes and interfaces. The infrastructure

preferably is reliable, available, scalable and manageable. Aggregation of trading partner or trader information from different marketplaces into a single super directory view of trading partners can be done at least three ways. Cache can be updated from trading partner information exchange between marketplaces in the infrastructure. Trading partner information can be replicated from one marketplace to another. Or, meta data about trading partners available in a single location or within each marketplace instance can refer the interested party to a location that makes available more complete data.

Fig. 1 depicts how the infrastructure adds value to the market portals by providing services 105 beyond acting as library for formats and protocols, interfaces, revenue-sharing agreements, privacy policies and the like. The services 105 provided include billing, traffic management, settlement of transactions, non-repudiation, a certificate authority, a commodity spot market, general security services, and a global directory for suppliers and items. The services may be implemented through an interface schema. A registry and repository 106 supports the infrastructure and services.

Fig. 2 illustrates an overall global trading infrastructure topology, with examples of five marketplaces. The global trading Web of infrastructure 201 comprises a global registry and repository 202, global community services 203, global trading services 204, and back office services 205. The names assigned to these groups of services are somewhat arbitrary. Examples of marketplaces in this figure include a regional MarketSite partner (BT) 211, a branded or vertical MarketSite such as Promus TM 212, CommerceOne's own MarketSite.net 213, a marketplace operated by BellSouth TM 214, and a second regional MarketSite partner (NTT). Buyers and suppliers are illustrated as being in communication with each marketplace. The existence of revenue-sharing agreements between various marketplaces is depicted by the revenue-sharing agreements arrow 220 between BT 211 and NTT 215.

Fig. 3 illustrates a network of market participants and market makers based on the use of business interface definitions, and supporting the trading of input and output documents specified according to such interface descriptions. The network includes a plurality of nodes 31-38 which are interconnected through a communication network such as the Internet 39, or other telecommunications or data communications network. Each of the nodes 31-39 consists of a computer, such as a portable computer, a desktop personal computer, a workstation, a network of systems, or other data processing

resources. The nodes include memory for storing the business interface definition, processors that execute transaction processes supporting commercial transactions with other nodes in the network, and computer programs which are executed by the processors in support of such services. In addition each of the nodes includes a network interface for providing for communication across the Internet 39, or the other communication network.

In the environment of Fig. 3, nodes 31, 32, 33, 34, 36 and 38 are designated market participants. Market participants include resources for consumers or suppliers of goods or services to be traded according to commercial transactions established according to the present invention.

In this example, nodes 35 and 37 are market maker nodes. The market maker nodes include resources for registering business interface definitions, called a BID registry. Participants are able to send documents to a market maker node, at which the document is identified and routed to an appropriate participant which has registered to receive such documents as input. The market maker also facilitates the commercial network by maintaining a repository of standard forms making up a common business library for use in building business interface definitions.

In this example, the market participant 38 is connected directly to the market maker 37, rather than through the Internet 39. This connection directly to the market maker illustrates that the configuration of the networks supporting commercial transactions can be very diverse, incorporating public networks such as the Internet 39, and private connections such as a local area network or a Point-to-Point connection as illustrated between nodes 37 and 38. Actual communication networks are quite diverse and suitable for use to establish commercial transaction networks according to the present invention.

Fig. 4 illustrates global trading infrastructure implementation options. The numbering here is parallel to Fig. 2. Both real and virtual document routing are illustrated. Documents 420 may be routed by the marketplaces through the global trading Web infrastructure 401. The document 421 may be exchanged peer-to-peer between marketplaces, supported by registry and repository information exchange between the global trading Web infrastructure 401 and the marketplaces 411 and 415.

The high-level architecture of the registry and repository is illustrated in Fig. 5. Interfaces are provided for communication through the Web 501 and exchange of

documents 502. A variety of repository services are illustrated. Information organization services 510 may include a default taxonomy, a taxonomy map and a commerce community schema. In general, services for browsing and searching 511 are provided. Information exchange services 512 may include publication or subscription of trading partners to a global training Web and publication or subscription to other marketplaces. Additional services 513 may include validation, transformation and management of documents passing through the document interface. The extensibility of the repository services 511 is by design. From an architectural perspective, one layer of the present invention provides for document storage to document mapping 520. Different document storage schemes may apply to different types of data. For instance X.500 may be used to store market trading partner information. An XML schema may map data that are stored in Java objects. A registry stores meta data 530. Standards adapted from Oasis and Dublin Core may be used for the registry. The existence of a them registry adds value to the market, for market participants, to services, documents, information items, trading partner relationships, routing, subscriptions and publications. The actual storage of data is implemented by data storage based on information types 540. Functions provided by this architecture across all layers 550 include security, reliability, availability, scalability and management functions.

Fig. 6 is a conceptual block diagram depicting entities and relationships in a commerce community schema. The network layer 601 is a root or global trading infrastructure, such as one provided by CommerceOne. The community layer 602 corresponds to regional or vertical partners in communication with the root of the network layer. Each of the partners relies on a marketplace definition layer 603 to support their instance of a regional or vertical marketplace. Details of the marketplace are provided at the marketplace specifics layer 604.

Fig. 7 is a hierarchical diagram of a data schema which implements an embodiment of the present invention. A top layer, a root 701 is in contact with various marketplaces 711-13. The root includes an owner, operator, technical contact and administrative contact. Each of the marketplaces include an nonowner, operator and technical and administrative contact. Any of the marketplaces may have a market information registry 721. The market information registry for CommerceOne, for instance, may be maintained on the server for MarketSite.net. The server for a single marketplace may maintain a registry security model including certificate authority

credentials 731, terms and conditions for transactions which express the "do's, don'ts and expirations" for transactions applicable throughout the marketplace 732, a market participants registry 733, core business documents written in CBL, XML schema language or in the form of a document guide 734 and logic for registration for core services 735.

The registry of market participants reflects both trading partners and services in their respective registries 741-42. Trading partners may include buyers, suppliers, service providers and others. A buyers registry pointer 751 is maintained as a unique universal resource name. Similarly, a suppliers registry contains supply registry pointers 752, a service provider registry contains pointers 753 and a organization registry contains pointers 754, all as universal resource names. The buyer registry pointers, for instance, may reference buying organizations including purchasing managers, purchasing administrators, system administrators and desktop requisitioners 771 and buying applications which may be identified by name and version number 772. Similarly, the supply registry pointers 752 may reference selling organizations including a sales manager, sales administrator, catalog manager, system administrators and IT managers. Supply registry pointers 752 also may reference selling applications which may be identified by name and version number 774.

The registry of services may include system services 761, business services 762, portal services 763, and community services 764. Each is the services may be registry through registry pointers as universal resource names. The nomenclature for these groups of services may alternatively be as depicted in Fig. 2: global community services, global trading services and back office services.

A further aspect of the present invention is an XML language schema which can be compiled by the CommerceOne Sox compiler into a series a Java objects. Superclasses and classes of objects are defined. The definitions below appear in alphabetical order, rather than any particular logical order. These definitions should be readily understood by those skilled in XML programming. The parsing of the schema can be illustrated in examples. The Sox definition for GtwCoreBusinessDocs.sox corresponds to core business documents 734 of Fig. 7. It is written in XML version 1.0. In is a "system" document type of a schema associated with the universal resource name x-commerceone:document:com:commerceone:xdk:xml:schema.dtd$1.0. The schema actually begins at the tag "schema". The uniform resource identifier for the schema

corresponds to the boldfaced routine name; in this example, it is x-commerceone:document:gtw:GtwCoreBusinessDocs.sox$1.0. Comments in this schema are indicated by tags. A comment begins with the tag "comment" and ends with the tag "/comment". Element types are used to expand this definition. An element type has a name which corresponds to the routine name and has a model. In a model, each data element can occur once only, as signified by "+", may be optional "?", or may appear many times "*". In this instance, a universal resource indicator is a pointer which occurs only once.

In each part of the example below, the **name** of the type is **bold** and its contents follow.

**GtwApplication.sox**

```
<?xml version="1.0"?>
<!DOCTYPE schema SYSTEM "urn:x-
commerceone:document:com:commerceone:xdk:xml:schema.dtd$1.0">

<schema uri="urn:x-commerceone:document:gtw:GtwApplication.sox$1.0">

<comment>
Copyright 1999 Commerce One Inc.
Created By:Bart Meltzer
Created Date: 10/1/99
Purpose:To define a basic Application in the GTW.
</comment>

  <comment>
  GtwApplication Can be extended with elementypes
  </comment>

  <elementtype name="GtwApplication">
    <model>
        <sequence>
      <element name="GtwApplicationName" type="string"/>
```

```
            <element name="GtwApplicationVersion" type="string"/>
            <element name="GtwApplicationManufacturer" type="string"/>
            <element name="GtwApplicationLiscenceOwner" type="string"/>
            <element name="GtwApplicationPrimaryProtocol" type="AppProtocol" occurs="?"/>
              </sequence>
        </model>
      </elementtype>

    <datatype name="AppProtocol">
          <enumeration datatype="NMTOKEN">
                <option>http</option>
                <option>https</option>
                <option>iiop</option>
                <option>mq</option>
                <option>xa</option>
                <option>smtp</option>
                <option>ftp</option>
                <option>edi</option>
                <option>fax</option>
          </enumeration>
    </datatype>

    </schema>
```

**GtwBusinessService.sox**

```
    <?xml version="1.0"?>
    <!DOCTYPE schema SYSTEM "urn:x-
    commerceone:document:com:commerceone:xdk:xml:schema.dtd$1.0">

    <schema uri="urn:x-commerceone:document:gtw:GtwBusinessService.sox$1.0">

    <comment>
    Copyright 1999 Commerce One Inc.
    Created By:Bart Meltzer
    Created Date: 10/1/99
    Purpose:To define business service in the GTW.
```

```
</comment>

<namespace prefix="GtwService" namespace="urn:x-
commerceone:document:gtw:GtwService.sox$1.0"/>

  <comment>
  GtwBusinessService Can be extended with elementypes, and is a wrapper on top of the GtwService
component.
  </comment>

  <elementtype name="GtwBusinessService">
    <model>
          <element name="ServiceData" type="GtwService" prefix="GtwService"/>
    </model>
  </elementtype>

</schema>
```

**GtwBuyer.sox**

```
<?xml version="1.0"?>
<!DOCTYPE schema SYSTEM "urn:x-
commerceone:document:com:commerceone:xdk:xml:schema.dtd$1.0">


<schema uri="urn:x-commerceone:document:gtw:GtwBuyer.sox$1.0">

<comment>
Copyright 1999 Commerce One Inc.
Created By:Bart Meltzer
Created Date: 10/1/99
Purpose:To define a basic Buyer in the GTW. It is defined here as a profile in a named MarketSite
Instance for the organization and the root of a descriptionof the applications they use which is not required
to be in the same Marketplace if they are already defined elsewhere on the GTW.
</comment>

<namespace prefix="GtwParticipants" namespace="urn:x-
commerceone:document:gtw:GtwParticipants.sox$1.0"/>
```

```
<namespace prefix="GtwTradingPartner" namespace="urn:x-
commerceone:document:gtw:GtwTradingPartner.sox$1.0"/>

  <comment>
  GtwBuyer Can be extended with elementypes, and is a type of GtwTradingPartber
  </comment>

  <elementtype name="GtwBuyer">
    <model>
      <sequence>
        <element name="GtwBuyerName" type="string"/>
        <element name="GtwBuyerID" type="guuid" prefix="GtwParticipants"/>
        <element name="GtwBuyerOrganizationProfileRoot" type="URI"/>
        <element name="GtwBuyerApplicationInformationRoot" type="URI"/>
      </sequence>
    </model>
  </elementtype>

</schema>
```

**GtwBuyerApplication.sox**

```
<?xml version="1.0"?>
<!DOCTYPE schema SYSTEM "urn:x-
commerceone:document:com:commerceone:xdk:xml:schema.dtd$1.0">


<schema uri="urn:x-commerceone:document:gtw:GtwBuyerApplication.sox$1.0">

<comment>
Copyright 1999 Commerce One Inc.
Created By:Bart Meltzer
Created Date: 10/1/99
Purpose:To define the Buyer Apllication in the GTW.
</comment>

<namespace prefix="GtwApplication" namespace="urn:x-
commerceone:document:gtw:GtwApplication.sox$1.0"/>
```

```
<comment>
GtwBuyerApplication Can be extended in this elementype and is a wrapper for GtwApplication component.
</comment>

<elementtype name="GtwBuyerApplication">
  <model>
    <element name="ApplicationInformation" type="GtwApplication" prefix="GtwApplication"/>
  </model>
</elementtype>

</schema>
```

**GtwBuyerOrganization.sox**

```
<?xml version="1.0"?>
<!DOCTYPE schema SYSTEM "urn:x-commerceone:document:com:commerceone:xdk:xml:schema.dtd$1.0">

<schema uri="urn:x-commerceone:document:gtw:GtwBuyerOrganization.sox$1.0">

<comment>
Copyright 1999 Commerce One Inc.
Created By:Bart Meltzer
Created Date: 10/1/99
Purpose:To define the Buyer Organization details of the Global Trading Web
</comment>

  <comment>
  Need to get copy of TP Registry Buyer Organization Profile and Buyer Individual Profile
  </comment>

</schema>
```

**GtwCommunityService.sox**

```
<?xml version="1.0"?>
<!DOCTYPE schema SYSTEM "urn:x-
commerceone:document:com:commerceone:xdk:xml:schema.dtd$1.0">

<schema uri="urn:x-commerceone:document:gtw:GtwCommunityService.sox$1.0">

<comment>
Copyright 1999 Commerce One Inc.
Created By:Bart Meltzer
Created Date: 10/1/99
Purpose:To define Community service in the GTW.
</comment>

<namespace prefix="GtwService" namespace="urn:x-
commerceone:document:gtw:GtwService.sox$1.0"/>

  <comment>
  GtwCommunityService Can be extended with elementtypes, and is a wrapper on top of the GtwService
component.
  </comment>

  <elementtype name="GtwCommunityService">
    <model>
            <element name="ServiceData" type="GtwService" prefix="GtwService"/>
    </model>
  </elementtype>

</schema>
```

**GtwCoreBusinessDocs.sox**

```
<?xml version="1.0"?>
<!DOCTYPE schema SYSTEM "urn:x-
commerceone:document:com:commerceone:xdk:xml:schema.dtd$1.0">
```

```
<schema uri="urn:x-commerceone:document:gtw:GtwCoreBusinessDocs.sox$1.0">

<comment>
Copyright 1999 Commerce One Inc.
Created By:Bart Meltzer
Created Date: 10/1/99
Purpose:To define a basic set of Business Documents in the GTW, and it can be used for the core
business documents in and MarketSite Instance.
</comment>

  <comment>
  GtwCoreBusinessDocs Can be extended with elementypes
  </comment>

  <elementtype name="GtwCoreBusinessDocs">
    <model>
      <element name="GtwCoreBusinessDocPointer" type="URI" occurs="+"/>
    </model>
  </elementtype>
</schema>
```

GtwCoreServices.sox

```
<?xml version="1.0"?>
<!DOCTYPE schema SYSTEM "urn:x-
commerceone:document:com:commerceone:xdk:xml:schema.dtd$1.0">

<schema uri="urn:x-commerceone:document:gtw:GtwCoreServices.sox$1.0">

<comment>
Copyright 1999 Commerce One Inc.
Created By:Bart Meltzer
Created Date: 10/1/99
Purpose:To define the basic Services in the GTW, and to define the core services in a MarketSite
Instance.
</comment>
```

```
<comment>
GtwCoreServices Can be extended with elementypes
</comment>

<elementtype name="GtwCoreServices">
  <model>
    <element name="GtwCoreServicePointer" type="URI" occurs="+"/>
  </model>
</elementtype>
</schema>
```

**GtwInfrastructureService.sox**

```
<?xml version="1.0"?>
<!DOCTYPE schema SYSTEM "urn:x-
commerceone:document:com:commerceone:xdk:xml:schema.dtd$1.0">


<schema uri="urn:x-commerceone:document:gtw:GtwInfrastructureService.sox$1.0">

<comment>
Copyright 1999 Commerce One Inc.
Created By:Bart Meltzer
Created Date: 10/1/99
Purpose:To define an infrastructure service in the GTW.
</comment>

<namespace prefix="GtwService" namespace="urn:x-
commerceone:document:gtw:GtwService.sox$1.0"/>

  <comment>
  GtwInfrastructureService Can be extended with elementypes, and is a wrapper on top of the
GtwService component.
  </comment>

  <elementtype name="GtwInfrastructureService">
    <model>
                <element name="ServiceData" type="GtwService" prefix="GtwService"/>
```

```
    </model>
  </elementtype>

</schema>
```

**GtwInterface_GtwRoot.sox**

```
<?xml version="1.0"?>
<!DOCTYPE schema SYSTEM "urn:x-
commerceone:document:com:commerceone:xdk:xml:schema.dtd$1.0">


<schema uri="urn:x-commerceone:document:gtw:GtwInterface_GtwRoot.sox$1.0">

<comment>
Copyright 1999 Commerce One Inc.
Created By:Bart Meltzer
Created Date: 10/2/99
Purpose: Interface to get GtwRoot information.
</comment>

<namespace prefix="GtwParticipants" namespace="urn:x-
commerceone:document:gtw:GtwParticipants.sox$1.0"/>
<namespace prefix="GtwService" namespace="urn:x-
commerceone:document:gtw:GtwService.sox$1.0"/>

  <comment>
  GtwInterface_GtwRoot Can be extended in this elementype by adding new services.
  </comment>

  <elementtype name="GtwInterface_GtwRoot">
    <model>
      <sequence>

        <element name="GtwRootRegistryOwner" type="Owner" prefix="GtwParticipants"/>
        <element name="GtwRootRegistryOperator" type="Operator" prefix="GtwParticipants"/>
        <element name="GtwRootRegistryTechnicalContact" type="TechnicalContact"
prefix="GtwParticipants"/>
```

<element name="GtwRootRegistryAdministrativeContact" type="AdministrativeContact" prefix="GtwParticipants"/>

<element name="GtwRootRegistry" type="URI"/>
<element name="GtwServiceRegistryRoot" type="URI"/>

<element name="GtwService_GtwRegistry_GetGtwRootRegistryRequestHandler" type="GtwService" prefix="GtwService"/>
<element name="GtwService_GtwRegistry_GetGtwServiceRegistryRequestHandler" type="GtwService" prefix="GtwService"/>
</sequence>
</model>
</elementtype>
</schema>

**GtwInterface_MarketInformationRegistry.sox**

<?xml version="1.0"?>
<!DOCTYPE schema SYSTEM "urn:x-commerceone:document:com:commerceone:xdk:xml:schema.dtd$1.0">

<schema uri="urn:x-commerceone:document:gtw:GtwInterface_MarketInformationRegistry.sox$1.0">

<comment>
Copyright 1999 Commerce One Inc.
Created By:Bart Meltzer
Created Date: 10/2/99
Purpose:To define the information service requirements for a GTW compliant Market Information Registry which is pointed to for every marketplace instance within an operation as defined by GtwOperation.sox. The information service requirements are in the form of:document exchanges that define: - general information - formats - functionality with interacitons. A GTW interface describes a set of information services that are available to interact with using XML documents.
</comment>

<namespace prefix="GtwParticipants" namespace="urn:x-commerceone:document:gtw:GtwParticipants.sox$1.0"/>

```
<namespace prefix="GtwService" namespace="urn:x-
commerceone:document:gtw:GtwService.sox$1.0"/>

  <comment>
  GtwInterface_MarketInformationRegistry Can be extended in this elementype by adding new services.
  </comment>

  <elementtype name="GtwInterface_MarketInformationRegistry">
    <model>
      <sequence>

        <element name="GtwMarketInformationRegistryOwner" type="Owner"
prefix="GtwParticipants"/>
        <element name="GtwMarketInformationRegistryOperator" type="Operator"
prefix="GtwParticipants"/>
        <element name="GtwMarketInformationRegistryTechnicalContact" type="TechnicalContact"
prefix="GtwParticipants"/>
        <element name="GtwMarketInformationRegistryAdministrativeContact"
type="AdministrativeContact" prefix="GtwParticipants"/>

                <element name="SecurityInformationRoot" type="URI"/>
                <element name="TermsAndConditionsRoot" type="URI"/>
                <element name="CoreBusinessDocumentsRoot" type="URI"/>
                <element name="CoreServicesRoot" type="URI"/>
                <element name="MarketParticipantRegistryRoot" type="URI"/>
                <element name="TradingPartnerDirectoryRoot" type="URI"/>
                <element name="ServiceRegistryRoot" type="URI"/>

                <element
name="GtwService_MarketInformationRegistry_GetMarketParticipantRegistryRequestHandler"
type="GtwService" prefix="GtwService"/>
                <element
name="GtwService_MarketInformationRegistry_GetTradingPartnerDirectoryRequestHandler"
type="GtwService" prefix="GtwService"/>

      </sequence>
    </model>
  </elementtype>
```

```
</schema>
```

**GtwInterface_MarketInstanceServiceRegistry.sox**

```
<?xml version="1.0"?>
<!DOCTYPE schema SYSTEM "urn:x-
commerceone:document:com:commerceone:xdk:xml:schema.dtd$1.0">


<schema uri="urn:x-
commerceone:document:gtw:GtwInterface_MarketInstanceServiceRegistry.sox$1.0">

<comment>
Copyright 1999 Commerce One Inc.
Created By:Bart Meltzer
Created Date: 10/2/99
Purpose: Interface to get GtwServices from a MarketSite Instance.
</comment>

<namespace prefix="GtwParticipants" namespace="urn:x-
commerceone:document:gtw:GtwParticipants.sox$1.0"/>
<namespace prefix="GtwService" namespace="urn:x-
commerceone:document:gtw:GtwService.sox$1.0"/>

  <comment>
  GtwInterface_MarketInstanceServiceRegistry Can be extended in this elementype by adding new
services.
  </comment>

  <elementtype name="GtwInterface_MarketInstanceServiceRegistry">
    <model>
      <sequence>

        <element name="GtwMarketInstanceServiceRegistryOwner" type="Owner"
prefix="GtwParticipants"/>
        <element name="GtwMarketInstanceServiceRegistryOperator" type="Operator"
prefix="GtwParticipants"/>
```

```
        <element name="GtwMarketInstanceServiceRegistryTechnicalContact"
type="TechnicalContact" prefix="GtwParticipants"/>
        <element name="GtwMarketInstanceServiceRegistryAdministrativeContact"
type="AdministrativeContact" prefix="GtwParticipants"/>

            <element name="MarketInstanceServiceRegistryRoot" type="URI"/>

            <element
name="GtwService_MarketInstanceServiceRegistry_GetServiceRegistryRootRequestHandler"
type="GtwService" prefix="GtwService"/>
        </sequence>
    </model>
  </elementtype>
</schema>
```

**GtwInterface_MarketInstanceTPRegistry.sox**

```
<?xml version="1.0"?>
<!DOCTYPE schema SYSTEM "urn:x-
commerceone:document:com:commerceone:xdk:xml:schema.dtd$1.0">

<schema uri="urn:x-commerceone:document:gtw:GtwInterface_MarketTPRegistry.sox$1.0">

<comment>
Copyright 1999 Commerce One Inc.
Created By:Bart Meltzer
Created Date: 10/2/99
Purpose: Interface to get Gtw Trading Partners from a MarketSite Instance.
</comment>

<namespace prefix="GtwParticipants" namespace="urn:x-
commerceone:document:gtw:GtwParticipants.sox$1.0"/>
<namespace prefix="GtwService" namespace="urn:x-
commerceone:document:gtw:GtwService.sox$1.0"/>

  <comment>
  GtwInterface_MarketTPRegistry Can be extended in this elementype by adding new services.
```

```
    </comment>

    <elementtype name="GtwInterface_MarketTPRegistry">
      <model>
        <sequence>

          <element name="GtwMarketTPRegistryOwner" type="Owner" prefix="GtwParticipants"/>
          <element name="GtwMarketTPRegistryOperator" type="Operator" prefix="GtwParticipants"/>
          <element name="GtwMarketTPRegistryTechnicalContact" type="TechnicalContact"
prefix="GtwParticipants"/>
          <element name="GtwMarketTPRegistryAdministrativeContact" type="AdministrativeContact"
prefix="GtwParticipants"/>

                <element name="MarketTPRegistryRoot" type="URI"/>

                <element name="GtwService_MarketTPRegistry_GetTPRegistryRootRequestHandler"
type="GtwService" prefix="GtwService"/>
        </sequence>
      </model>
    </elementtype>
</schema>
```

**GtwInterface_MarketParticipantRegistry.sox**

```
<?xml version="1.0"?>
<!DOCTYPE schema SYSTEM "urn:x-
commerceone:document:com:commerceone:xdk:xml:schema.dtd$1.0">


<schema uri="urn:x-commerceone:document:gtw:GtwInterface_MarketParticipantRegistry.sox$1.0">

<comment>
Copyright 1999 Commerce One Inc.
Created By:Bart Meltzer
Created Date: 10/2/99
Purpose: Interface to get GtwParticipants informaiton from a TP Registry.
</comment>
```

```
<namespace prefix="GtwParticipants" namespace="urn:x-
commerceone:document:gtw:GtwParticipants.sox$1.0"/>
<namespace prefix="GtwService" namespace="urn:x-
commerceone:document:gtw:GtwService.sox$1.0"/>

  <comment>
  GtwInterface_MarketParticipantRegistry Can be extended in this elementype by adding new services.
  </comment>

  <elementtype name="GtwInterface_MarketParticpantRegistry">
    <model>
      <sequence>

        <element name="GtwMarketParticipantRegistryOwner" type="Owner"
prefix="GtwParticipants"/>
        <element name="GtwMarketParticipantRegistryOperator" type="Operator"
prefix="GtwParticipants"/>
        <element name="GtwMarketParticipantRegistryTechnicalContact" type="TechnicalContact"
prefix="GtwParticipants"/>
        <element name="GtwMarketParticipantRegistryAdministrativeContact"
type="AdministrativeContact" prefix="GtwParticipants"/>

            <element name="TPRegistryRoot" type="URI"/>
            <element name="ServiceRegistryRoot" type="URI"/>

            <element
name="GtwService_MarketInformationRegistry_GetTPRegistryRootRequestHandler"
type="GtwService" prefix="GtwService"/>
            <element
name="GtwService_MarketInformationRegistry_GetServiceRegistryRootRequestHandler"
type="GtwService" prefix="GtwService"/>

      </sequence>
    </model>
  </elementtype>
</schema>
```

**GtwOperation.sox**

```
<?xml version="1.0"?>
<!DOCTYPE schema SYSTEM "urn:x-
commerceone:document:com:commerceone:xdk:xml:schema.dtd$1.0">

<schema uri="urn:x-commerceone:document:gtw:GtwOperation.sox$1.0">

<comment>
Copyright 1999 Commerce One Inc.
Created By:Bart Meltzer
Created Date: 10/1/99
Purpose:To define the operations in the Global Trading Web. An operation is defined by the Marketplaces
it Operates.
</comment>

<namespace prefix="GtwParticipants" namespace="urn:x-
commerceone:document:gtw:GtwParticipants.sox$1.0"/>

  <comment>
  GtwOperation Can be extended in this elementype
  </comment>

  <elementtype name="GtwOperation">
    <model>
      <sequence>

        <element name="GtwOperationOwner" type="Owner" prefix="GtwParticipants"/>
        <element name="GtwOperationOperator" type="Operator" prefix="GtwParticipants"/>
        <element name="GtwOperationTechnicalContact" type="TechnicalContact"
prefix="GtwParticipants"/>
        <element name="GtwOperationAdministrativeContact" type="AdministrativeContact"
prefix="GtwParticipants"/>

        <element name="MarketSiteOperation" type="MarketSiteInstance" occurs="+"/>

        <element name="ecoOperation" type="ecoInstance" occurs="*"/>
        <element name="OtherOperation" type="OtherInstance" occurs="*"/>
```

```
            </sequence>
          </model>
        </elementtype>

        <elementtype name="MarketSiteInstance">
            <empty/>
          <attdef name="Name" datatype="string">
            <required/>
          </attdef>
          <attdef name="MarketSiteNetworkRoot" datatype="URI">
            <required/>
          </attdef>
          <attdef name="Interface_MarketInformationRegistry" datatype="URI">
            <required/>
          </attdef>
          <attdef name="Interface_MarketParticipantRegistry" datatype="URI">
            <required/>
          </attdef>
          <attdef name="Interface_TradingPartnerDirectory" datatype="URI">
            <required/>
          </attdef>
        </elementtype>

        <elementtype name="ecoInstance">
            <empty/>
          <attdef name="ecoMarketplaceRoot" datatype="URI">
            <required/>
          </attdef>
          <attdef name="Name" datatype="string">
            <required/>
          </attdef>
          <attdef name="Interface_MarketInformationRegistry" datatype="URI">
            <required/>
          </attdef>
          <attdef name="Interface_TradingPartnerDirectory" datatype="URI">
            <required/>
          </attdef>
        </elementtype>
```

```
<elementtype name="OtherInstance">
    <empty/>
  <attdef name="OtherMarketplaceRoot" datatype="URI">
    <required/>
  </attdef>
  <attdef name="Name" datatype="string">
    <required/>
  </attdef>
  <attdef name="Interface_MarketInformationRegistry" datatype="URI">
    <required/>
  </attdef>
  <attdef name="Interface_TradingPartnerDirectory" datatype="URI">
    <required/>
  </attdef>
</elementtype>

</schema>
```

**GtwOther.sox**

```
<?xml version="1.0"?>
<!DOCTYPE schema SYSTEM "urn:x-
commerceone:document:com:commerceone:xdk:xml:schema.dtd$1.0">


<schema uri="urn:x-commerceone:document:gtw:GtwOtherEntity.sox$1.0">

<comment>
Copyright 1999 Commerce One Inc.
Created By:Bart Meltzer
Created Date: 10/1/99
Purpose:To define a basic OtherEntity in the GTW. It is defined here as a profile in a named MarketSite
Instance for the organization and the root of a descriptionof the applications they use which is not required
to be in the same Marketplace if they are already defined elsewhere on the GTW.
</comment>
```

```
<namespace prefix="GtwParticipants" namespace="urn:x-
commerceone:document:gtw:GtwParticipants.sox$1.0"/>
<namespace prefix="GtwTradingPartner" namespace="urn:x-
commerceone:document:gtw:GtwTradingPartner.sox$1.0"/>

  <comment>
  GtwOtherEntity Can be extended with elementypes, and is a type of GtwTradingPartner
  </comment>

  <elementtype name="GtwOtherEntity">
    <model>
      <sequence>
        <element name="GtwOtherEntityName" type="string"/>
        <element name="GtwOtherEntityID" type="guuid" prefix="GtwParticipants"/>
        <element name="GtwOtherEntityOrganizationProfileRoot" type="URI"/>
        <element name="GtwOtherEntityInformationRoot" type="URI"/>
      </sequence>
    </model>
  </elementtype>

</schema>
```

**GtwParticipants.sox**

```
<?xml version="1.0"?>
<!DOCTYPE schema SYSTEM "urn:x-
commerceone:document:com:commerceone:xdk:xml:schema.dtd$1.0">


<schema uri="urn:x-commerceone:document:gtw:GtwParticipants.sox$1.0">

<comment>
Copyright 1999 Commerce One Inc.
Created By:Bart Meltzer
Created Date: 10/1/99
Purpose:To define participants in the Global Trading Web
</comment>
```

```
<comment>
GtwParticipants Can be extended with elementypes
</comment>

<elementtype name="Owner">
    <model>
        <element name="OwnerName" type="GtwActor"/>
    </model>
    <attdef name="OwnerID" datatype="guuid">
        <required/>
    </attdef>
</elementtype>

<elementtype name="Operator">
    <model>
        <element name="OperatorName" type="GtwActor"/>
    </model>
    <attdef name="OperatorID" datatype="guuid">
        <required/>
    </attdef>
</elementtype>

<elementtype name="TechnicalContact">
    <model>
        <element name="TechnicalContactName" type="GtwActor"/>
    </model>
    <attdef name="TechnicalContactID" datatype="guuid">
        <required/>
    </attdef>
</elementtype>

<elementtype name="AdministrativeContact">
    <model>
        <element name="AdministrativeContactName" type="GtwActor"/>
    </model>
    <attdef name="AdminstrativeContactID" datatype="guuid">
        <required/>
    </attdef>
```

```
    </elementtype>

    <elementtype name="GtwActor">
        <model>
            <string/>
        </model>
    </elementtype>

    <datatype name="guuid">
      <scalar datatype="int"/>
    </datatype>

</schema>
```

**GtwPortalService.sox**

```
<?xml version="1.0"?>
<!DOCTYPE schema SYSTEM "urn:x-
commerceone:document:com:commerceone:xdk:xml:schema.dtd$1.0">

<schema uri="urn:x-commerceone:document:gtw:GtwPortalService.sox$1.0">

<comment>
Copyright 1999 Commerce One Inc.
Created By:Bart Meltzer
Created Date: 10/1/99
Purpose:To define portal service in the GTW.
</comment>

<namespace prefix="GtwService" namespace="urn:x-
commerceone:document:gtw:GtwService.sox$1.0"/>

  <comment>
  GtwPortalService Can be extended with elementypes, and is a wrapper on top of the GtwService
component.
  </comment>
```

```
    <elementtype name="GtwPortalService">
      <model>
                <element name="ServiceData" type="GtwService" prefix="GtwService"/>
      </model>
    </elementtype>

</schema>
```

**GtwQos.sox**

```
<?xml version="1.0"?>
<!DOCTYPE schema SYSTEM "urn:x-
commerceone:document:com:commerceone:xdk:xml:schema.dtd$1.0">


<schema uri="urn:x-commerceone:document:gtw:GtwQOS.sox$1.0">

<comment>
Copyright 1999 Commerce One Inc.
Created By:Bart Meltzer
Created Date: 10/1/99
Purpose:To define the basic Quality of Service Statements that can be disclosed int he GTW.
</comment>

  <comment>
  GtwQOS Can be extended in this elementype

  Needs to be changed from any string to an elelemttype that inforces the policy for how QOS
informaiton should be communicated in the GTW.
  </comment>

  <elementtype name="GtwQOS">
    <model>
          <element name="GtwQOSStatement" type="string"/>
    </model>
  </elementtype>
</schema>
```

**GtwRoot.sox**

```
<?xml version="1.0"?>
<!DOCTYPE schema SYSTEM "urn:x-
commerceone:document:com:commerceone:xdk:xml:schema.dtd$1.0">

<schema uri="urn:x-commerceone:document:gtw:GtwRoot.sox$1.0">

<comment>
Copyright 1999 Commerce One Inc.
Created By:Bart Meltzer
Created Date: 10/1/99
Purpose:To define the root node of the Global Trading Web
</comment>

<namespace prefix="GtwParticipants" namespace="urn:x-
commerceone:document:gtw:GtwParticipants.sox$1.0"/>

  <comment>
  GtwRoot Can be extended in this elementype
  </comment>

  <elementtype name="GtwRoot">
    <model>
      <sequence>
        <element name="GtwRootOwner" type="Owner" prefix="GtwParticipants"/>
<element name="GtwRootOperator" type="Operator" prefix="GtwParticipants"/>
        <element name="GtwRootTechnicalContact" type="TechnicalContact"
prefix="GtwParticipants"/>
        <element name="GtwRootAdministrativeContact" type="AdministrativeContact"
prefix="GtwParticipants"/>

        <element name="GtwOperation" type="URI" occurs="+"/>
      </sequence>
    </model>
  </elementtype>
```

```
</schema>
```

**GtwSecurity.sox**

```
<?xml version="1.0"?>
<!DOCTYPE schema SYSTEM "urn:x-
commerceone:document:com:commerceone:xdk:xml:schema.dtd$1.0">

<schema uri="urn:x-commerceone:document:gtw:GtwSecurity.sox$1.0">

<comment>
Copyright 1999 Commerce One Inc.
Created By:Bart Meltzer
Created Date: 10/1/99
Purpose:To define the security in the Global Trading Web.
</comment>

<namespace prefix="GtwParticipants" namespace="urn:x-
commerceone:document:gtw:GtwParticipants.sox$1.0"/>

  <comment>
  GtwSecurity Can be extended in this elementype
  </comment>

  <elementtype name="GtwSecurity">
    <model>
        <sequence>
            <element name="CertifcateAuthority" type="URI"/>
            <element name="RegistrationAuthority" type="URI"/>
        </sequence>
    </model>
  </elementtype>
</schema>
```

**GtwService.sox**

```
<?xml version="1.0"?>
```

```
<!DOCTYPE schema SYSTEM "urn:x-
commerceone:document:com:commerceone:xdk:xml:schema.dtd$1.0">

<schema uri="urn:x-commerceone:document:gtw:GtwService.sox$1.0">

<comment>
Copyright 1999 Commerce One Inc.
Created By:Bart Meltzer
Created Date: 10/1/99
Purpose:To define a basic service in the GTW.
</comment>

<namespace prefix="GtwParticipants" namespace="urn:x-
commerceone:document:gtw:GtwParticipants.sox$1.0"/>

  <comment>
  GtwService Can be extended with elementypes
  </comment>

  <elementtype name="GtwService">
    <model>
        <sequence>
      <element name="GtwServiceOwner" type="Owner" prefix="GtwParticipants"/>
            <element name="GtwServiceOperator" type="Operator" prefix="GtwParticipants"/>
            <element name="GtwServiceTechnicalContact" type="TechnicalContact"
prefix="GtwParticipants"/>
      <element name="GtwServiceAdministrativeContact" type="AdministrativeContact"
prefix="GtwParticipants"/>

            <element name="GtwInteraction" type="DocumentExchange" occurs="+"/>
        </sequence>
    </model>
  </elementtype>

  <elementtype name="DocumentExchange">
      <model>
            <sequence>
```

```
                    <element name="DocumentExchangeName" type="string"/>
                    <element name="DocumentExchangeType" type="docxchangetype"/>
                    <element name="DocumentExchangeProtocol" type="docxchangeprotocol"/>
                    <element name="DocumentExchangeLocation" type="URI"/>
                    <element name="DocumentExchangeQOS" type="URI"/>
                    <element name="DocumentExchangeInputDoc" type="URI"/>
                    <element name="DocumentExchangeOutputDoc" type="URI"/>
                    <element name="DocumentExchangeErrorDoc" type="URI"/>
                    <element name="DocumentExchangeCancelDoc" type="URI"/>
                    <element name="DocumentExchangeInquiryDoc" type="URI"/>
                </sequence>
            </model>
        </elementtype>

    <datatype name="docxchangetype">
        <enumeration datatype="NMTOKEN">
            <option>request</option>
            <option>response</option>
            <option>error</option>
            <option>cancel</option>
            <option>inquiry</option>
        </enumeration>
    </datatype>

    <datatype name="docxchangeprotocol">
        <enumeration datatype="NMTOKEN">
            <option>http</option>
            <option>https</option>
            <option>iiop</option>
            <option>mq</option>
            <option>xa</option>
            <option>smtp</option>
            <option>ftp</option>
            <option>edi</option>
            <option>fax</option>
        </enumeration>
    </datatype>
```

```
</schema>
```

**GtwServiceProvider.sox**

```
<?xml version="1.0"?>
<!DOCTYPE schema SYSTEM "urn:x-
commerceone:document:com:commerceone:xdk:xml:schema.dtd$1.0">


<schema uri="urn:x-commerceone:document:gtw:GtwServiceProvider.sox$1.0">

<comment>
Copyright 1999 Commerce One Inc.
Created By:Bart Meltzer
Created Date: 10/1/99
Purpose:To define a basic ServiceProvider in the GTW. It is defined here as a profile in a named
MarketSite Instance for the organization and the root of a descriptionof the applications they use which is
not required to be in the same Marketplace if they are already defined elsewhere on the GTW.
</comment>

<namespace prefix="GtwParticipants" namespace="urn:x-
commerceone:document:gtw:GtwParticipants.sox$1.0"/>
<namespace prefix="GtwTradingPartner" namespace="urn:x-
commerceone:document:gtw:GtwTradingPartner.sox$1.0"/>

  <comment>
  GtwServiceProvider Can be extended with elementypes, and is a type of GtwTradingPartber
  </comment>

  <elementtype name="GtwServiceProvider">
    <model>
        <sequence>
            <element name="GtwServiceProviderName" type="string"/>
            <element name="GtwServiceProviderID" type="guuid" prefix="GtwParticipants"/>
            <element name="GtwServiceProviderOrganizationProfileRoot" type="URI"/>
            <element name="GtwServiceProviderServiceInformationRoot" type="URI"/>
        </sequence>
    </model>
```

```
</elementtype>

</schema>
```

**GtwSupplier.sox**

```
<?xml version="1.0"?>
<!DOCTYPE schema SYSTEM "urn:x-
commerceone:document:com:commerceone:xdk:xml:schema.dtd$1.0">


<schema uri="urn:x-commerceone:document:gtw:GtwSupplier.sox$1.0">

<comment>
Copyright 1999 Commerce One Inc.
Created By:Bart Meltzer
Created Date: 10/1/99
Purpose:To define a basic Supplier in the GTW. It is defined here as a profile in a named MarketSite
Instance for the organization and the root of a descriptionof the applications they use which is not required
to be in the same Marketplace if they are already defined elsewhere on the GTW.
</comment>

<namespace prefix="GtwParticipants" namespace="urn:x-
commerceone:document:gtw:GtwParticipants.sox$1.0"/>
<namespace prefix="GtwTradingPartner" namespace="urn:x-
commerceone:document:gtw:GtwTradingPartner.sox$1.0"/>

  <comment>
  GtwSupplier Can be extended with elementypes, and is a type of GtwTradingPartber
  </comment>

  <elementtype name="GtwSupplier">
    <model>
        <sequence>
            <element name="GtwSupplierName" type="string"/>
            <element name="GtwSupplierID" type="guuid" prefix="GtwParticipants"/>
            <element name="GtwSupplierOrganizationProfileRoot" type="URI"/>
            <element name="GtwSupplierApplicationInformationRoot" type="URI"/>
```

```
            </sequence>
        </model>
    </elementtype>

</schema>
```

**GtwSupplierApplication.sox**

```
<?xml version="1.0"?>
<!DOCTYPE schema SYSTEM "urn:x-commerceone:document:com:commerceone:xdk:xml:schema.dtd$1.0">

<schema uri="urn:x-commerceone:document:gtw:GtwSupplierApplication.sox$1.0">

<comment>
Copyright 1999 Commerce One Inc.
Created By:Bart Meltzer
Created Date: 10/1/99
Purpose:To define the Supplier Apllication in the GTW.
</comment>

<namespace prefix="GtwApplication" namespace="urn:x-commerceone:document:gtw:GtwApplication.sox$1.0"/>

  <comment>
  GtwSupplierApplication Can be extended in this elementype and is a wrapper for GtwApplication component.
  </comment>

  <elementtype name="GtwSupplierApplication">
    <model>
            <element name="ApplicationInformation" type="GtwApplication" prefix="GtwApplication"/>
    </model>
  </elementtype>

</schema>
```

**GtwSupplierOrganization.sox**

```
<?xml version="1.0"?>
<!DOCTYPE schema SYSTEM "urn:x-
commerceone:document:com:commerceone:xdk:xml:schema.dtd$1.0">

<schema uri="urn:x-commerceone:document:gtw:GtwSupplierOrganization.sox$1.0">

<comment>
Copyright 1999 Commerce One Inc.
Created By:Bart Meltzer
Created Date: 10/1/99
Purpose:To define the Supplier Organization details of the Global Trading Web
</comment>

  <comment>
  Need to get copy of TP Registry Supplier Organization Profile and Supplier Individual Profile
  </comment>

</schema>
```

**GtwTermsAndConditions.sox**

```
<?xml version="1.0"?>
<!DOCTYPE schema SYSTEM "urn:x-
commerceone:document:com:commerceone:xdk:xml:schema.dtd$1.0">

<schema uri="urn:x-commerceone:document:gtw:GtwTermsAndConditions.sox$1.0">

<comment>
Copyright 1999 Commerce One Inc.
Created By:Bart Meltzer
Created Date: 10/1/99
Purpose:To define the TermsAndConditions in the Global Trading Web. An operation defines terms and
conditions in each Marketsite that are consistent with GTW Policy.
```

```
</comment>

<namespace prefix="GtwParticipants" namespace="urn:x-
commerceone:document:gtw:GtwParticipants.sox$1.0"/>

  <comment>
  GtwTermsAndConditions Can be extended in this elementype
  Terms and Conditions needs to be structured beyond what is here now.
  </comment>

  <elementtype name="GtwTermsAndConditions">
    <model>
        <string/>
    </model>
  </elementtype>

</schema>
```

**GtwTradingPartner.sox**

```
<?xml version="1.0"?>
<!DOCTYPE schema SYSTEM "urn:x-
commerceone:document:com:commerceone:xdk:xml:schema.dtd$1.0">

<schema uri="urn:x-commerceone:document:gtw:GtwTradingPartner.sox$1.0">

<comment>
Copyright 1999 Commerce One Inc.
Created By:Bart Meltzer
Created Date: 10/1/99
Purpose:To define a basic Trading Partner in the GTW. It is defined here as a profile in a named
MarketSite Instance.
</comment>

<namespace prefix="GtwParticipants" namespace="urn:x-
commerceone:document:gtw:GtwParticipants.sox$1.0"/>
```

<comment>
GtwTradingPartner Can be extended with elementypes, but should never redefine the Marketplace profile established with the guuid. It is ok to extend beyond the Marketplace profile for GTW purposes. The GtwTradingPartner URI is meant to be a query back to the Trading Partner Registry in the GTW where the Trading Partner is defined.
GtwTradingPartnerName needs to be TPName instead of string
GtwTradingPartnerID needs to be MPID which is itself of type guuid
</comment>

```
<elementtype name="GtwTradingPartner">
  <model>
    <sequence>
      <element name="GtwTradingPartnerName" type="string"/>
      <element name="GtwTradingPartnerID" type="guuid" prefix="GtwParticipants"/>
      <element name="GtwTradingPartnerProfilePointer" type="URI"/>
    </sequence>
  </model>
</elementtype>
</schema>
```

**GtwWireFormat.sox**

```
<?xml version="1.0"?>
<!DOCTYPE schema SYSTEM "urn:x-
commerceone:document:com:commerceone:xdk:xml:schema.dtd$1.0">

<schema uri="urn:x-commerceone:document:gtw:GtwBuyerOrganization.sox$1.0">
```

<comment>
Copyright 1999 Commerce One Inc.
Created By:Bart Meltzer
Created Date: 10/1/99
Purpose:To define the Wire Format details of the Global Trading Web
</comment>

<comment>

Need to get copy of envelope specification for MIME supported in MarketSite 3.0 as well as transmitter characteristics.

</comment>

</schema>

In this schema, GtwPortalService.sox is a specialization of GtwService.sox. In the Java programming language, GtwService.Sox may represent a superclass which includes the class GtwPortalService.sox. In this schema, a namespace prefix is used to shorten references. A universal resource name is associated with the namespace prefix. The model for the element type "GtwService" includes an owner, operator, technical contact and administrative contact as depicted in Fig. 7. In addition, an element type for document exchange is defined. Data types used in the document exchange element type are also defined as part of the schema. When this schema has been defined, it can be specialized, as in "GtwPortalService.sox". With these examples, one having skill in XML programming and reference to Fig. 7 will be able to follow the schema in Fig. 8 which is one embodiment of the present invention.

Fig. 8 is a heuristic diagram of nested structures in a business interface definition BID which is established for market participants in the network according to the present invention. The business interface definition illustrated in Fig. 8 is a data structure that consists of logic structures and storage units arranged according to a formal definition of a document structure, such as a XML document type definition DTD. The structure of Fig. 8 includes a first logic structure 800 for identifying a party. Associated with the logic structure 800 are nested logic structures for carrying the name 801, the physical address 802, the network address or location 803, and a set of transactions for a service 804. For each transaction in the service set, an interface definition is provided, including the transaction BID 805, the transaction BID 806, and the transaction BID 807. Within each transaction BID, such as transaction BID 805, logical structures are provided for including a name 808, a location on the network at which the service is performed 809, the operations performed by the service 810, and a set of input documents indicated by the tag 811. Also, the service BID 805 includes a set of output documents indicated by the tag 812. The set of input documents 811 includes a business interface definition for each input document for which the services are designed to respond, including input

document business interface definitions 813, 814, and 815. Each business interface definition for an input document includes a name 816, a location on the network at which a description of the document can be found 817, and the modules to be carried in the document as indicated by the field 818. In a similar manner, the output document set 812 includes interface definitions for output documents including the output document BID 819, output document BID 820, and output document BID 821. For each output document BID, a name 822, a location on the network or elsewhere 823, and the modules of the document 824 are specified. The business interface definition for the participant as illustrated in Fig. 8 includes actual definitions of a logic structures to be utilized for the input and output documents of the respective services, or pointers or other references to locations at which these definitions can be found.

In the system of this example, the document of Fig. 8 is specified in an XML document type definition DTD, although other document definition architectures could be used, and includes interpretation information for the logical structures used in interpretation of instances of the documents. In addition, each of the transaction BIDs, input document BIDs and output document BIDs are specified according to an XML document type definitions. The XML type document is an example of a system based on parsed data that includes mark-up data and character data. Mark-up data identifies logical structures within the document and sets of character data identify the content of the logical structures. In addition, unparsed data can be carried in the document for a variety of purposes. See for example the specification of the *Extensible Mark-up Language XML 1.0* REC-XML-19980210 published by the WC3 XML Working Group at WWW.W3.ORG/TR/1998/REC-XML-19980210.

Thus in an exemplary system, participant nodes in the network establish virtual enterprises by interconnecting business systems and services with XML encoded documents that businesses accept and generate. For example, the business interface definition of a particular service establishes that if a document matching the BID of a request for a catalog entry is received, then a document matching a BID of a catalog entry will be returned. Also, if a document matching the BID of a purchase order is received, and it is acceptable to the receiving terminal, a document matching the BID of an invoice will be returned. The nodes in the network process the XML documents before they enter the local business system, which is established according to the variant transaction processing architecture of any given system in the network. Thus, the system

unpacks sets of related documents, such as MIME-encoded sets of XML documents, parses them to create a stream of "mark-up messages". The messages are routed to the appropriate applications and services using for example an event listener model like that described below.

The documents exchanged between business services are encoded using an XML language built from a repository of building blocks (a common business language) from which more complex document definitions may be created. The repository stores modules of interpretation information that are focused on the functions and information common to business domains, including business description primitives like companies, services and products; business forms like catalogs, purchase orders and invoices; standard measurements, like time, date, location; classification codes and the like providing interpretation information for logical structures in the XML documents.

The business interface definition is a higher level document that acts as a schema used for designing interfaces that trade documents according to the present invention. Thus the business interface definition bridges the gap between the documents specified according to XML and the programs which execute on the front end of the transaction processing services at particular nodes. Such front ends are implemented by JAVA virtual machines, or by other common architectures providing for interconnection of systems across a network. Thus, the business interface definition provides a technique by which a transaction protocol is programmed using the business interface definition document. The program for the protocol of the transaction is established by a detailed formal specification of a document type.

An example business interface definition BID based on a market participant document which conforms to an XML format is provided below. The market participant DTD groups business information about market participants, associating contact and address information with a description of services and financial information. This business information is composed of names, codes, addresses, a dedicated taxonomic mechanism for describing business organization, and a pointer to terms of business. In addition, the services identified by the market participant DTD will specify the input and output documents which that participant is expected respond to and produce. Thus, documents which define schema using an exemplary common business language for a market participant DTD, a service DTD, and a transaction document DTD specified in XML with explanatory comments follow:

**Market Participant Sample**

```
<!DOCTYPE SCHEMA SYSTEM "bidl.dtd">
<SCHEMA>
<H1>Market Participant Sample BID</H1>
<META
WHO.OWNS="Veo Systems"        WHO.COPYRIGHT="Veo Systems"
WHEN.COPYRIGHT="1998"         DESCRIPTION="Sample BID"
WHO.CREATED="*"               WHEN.CREATED="*"
WHAT.VERSION="*"              WHO.MODIFIED="*"
WHEN.MODIFIED="*"             WHEN.EFFECTIVE="*"
WHEN.EXPIRES="*"              WHO.EFFECTIVE="*"
WHO.EXPIRES="*">
</META>

<PROLOG>
<XMLDECL STANDALONE="no"></XMLDECL>
<DOCTYPE NAME="market.participant">
<SYSTEM>markpart.dtd</SYSTEM></DOCTYPE>
</PROLOG>

<DTD NAME="markpart.dtd">
<H2>Market Participant</H2>
<H3>Market Participant</H3>
<ELEMENTTYPE NAME="market.participant">
<EXPLAIN><TITLE>A Market Participant</TITLE>
<SYNOPSIS>A business or person and its service interfaces.</SYNOPSIS>
<P>A market participant is a document definition that is created to describe a business and at least one
person with an email address, and it presents a set of pointers to service interfaces located on the network.
In this example, the pointers have been resolved and the complete BID is presented
here.</P></EXPLAIN>
<MODEL><CHOICE>
<ELEMENT NAME="business"></ELEMENT>
<ELEMENT NAME="person"></ELEMENT>
</CHOICE></MODEL></ELEMENTTYPE>

<H3>Party Prototype</H3>
<PROTOTYPE NAME="party">
```

```
<EXPLAIN><TITLE>The Party Prototype</TITLE></EXPLAIN>
<MODEL><SEQUENCE>
<ELEMENT NAME="party.name" OCCURS="+"></ELEMENT>
<ELEMENT NAME="address.set"></ELEMENT>
</SEQUENCE></MODEL>
</PROTOTYPE>

<H3>Party Types</H3>
<ELEMENTTYPE NAME="business">
<EXPLAIN><TITLE>A Business</TITLE>
<SYNOPSIS>A business (party) with a business number attribute.</SYNOPSIS>
<P>This element inherits the content model of the party prototype and adds a business number attribute,
which serves as a key for database lookup. The business number may be used as a cross-reference to/from
customer id, credit limits, contacts lists, etc.</P></EXPLAIN>
<EXTENDS HREF="party">
<ATTDEF NAME="business.number"><REQUIRED></REQUIRED></ATTDEF>
</EXTENDS>
</ELEMENTTYPE>

<H3>Person Name</H3>
<ELEMENTTYPE NAME="person">
<EXPLAIN><TITLE>A Person</TITLE></EXPLAIN>
<EXTENDS HREF="party">
<ATTDEF NAME="SSN"><IMPLIED></IMPLIED></ATTDEF>
</EXTENDS>
</ELEMENTTYPE>

<H3>Party Name</H3>
<ELEMENTTYPE NAME="party.name">
<EXPLAIN><TITLE>A Party's Name</TITLE>
<SYNOPSIS>A party's name in a string of character.</SYNOPSIS></EXPLAIN>
<MODEL><STRING></STRING></MODEL>
</ELEMENTTYPE>

<H3>Address Set</H3>
<ELEMENTTYPE NAME="address.set">
<MODEL><SEQUENCE>
<ELEMENT NAME="address.physical"></ELEMENT>
```

```
<ELEMENT NAME="telephone" OCCURS="*"></ELEMENT>
<ELEMENT NAME="fax" OCCURS="*"></ELEMENT>
<ELEMENT NAME="email" OCCURS="*"></ELEMENT>
<ELEMENT NAME="internet" OCCURS="*"></ELEMENT>
</SEQUENCE></MODEL>
</ELEMENTTYPE>

<H3>Physical Address</H3>
<ELEMENTTYPE NAME="address.physical">
<EXPLAIN><TITLE>Physical Address</TITLE>
<SYNOPSIS>The street address, city, state, and zip code.</SYNOPSIS></EXPLAIN>
<MODEL><SEQUENCE>
<ELEMENT NAME="street"></ELEMENT>
<ELEMENT NAME="city"></ELEMENT>
<ELEMENT NAME="state"></ELEMENT>
<ELEMENT NAME="postcode" OCCURS="?"></ELEMENT>
<ELEMENT NAME="country"></ELEMENT>
</SEQUENCE></MODEL>
</ELEMENTTYPE>

<H3>Street</H3>
<ELEMENTTYPE NAME="street">
<EXPLAIN><TITLE>Street Address</TITLE>
<SYNOPSIS>Street or postal address.</SYNOPSIS></EXPLAIN>
<MODEL><STRING></STRING></MODEL>
</ELEMENTTYPE>

<H3>City</H3>
<ELEMENTTYPE NAME="city">
<EXPLAIN><TITLE>City Name or Code</TITLE>
<P>The city name or code is a string that contains sufficient information to identify a city within a
designated state.</P>
</EXPLAIN>
<MODEL><STRING></STRING></MODEL>
</ELEMENTTYPE>

<H3>State</H3>
<ELEMENTTYPE NAME="state">
```

<EXPLAIN><TITLE>State, Province or Prefecture Name or Code</TITLE>
<P>The state name or code contains sufficient information to identify a state within a designated country.</P></EXPLAIN>
<MODEL><STRING DATATYPE="COUNTRY.US.SUBENTITY"></STRING></MODEL>
</ELEMENTTYPE>

<H3>Postal Code</H3>
<ELEMENTTYPE NAME="postcode">
<EXPLAIN><TITLE>Postal Code</TITLE>
<P>A postal code is an alphanumeric code, designated by an appropriate postal authority, that is used to identify a location or region within the jurisdiction of that postal authority. Postal authorities include designated national postal authorities.</P></EXPLAIN>
<MODEL><STRING DATATYPE="string"></STRING></MODEL>
</ELEMENTTYPE>

<H3>Country</H3>
<ELEMENTTYPE NAME="country">
<EXPLAIN><TITLE>Country Code</TITLE>
<P>A country code is a two-letter code, designated by ISO, that is used to uniquely identify a country.</P></EXPLAIN>
<MODEL><STRING DATATYPE="country"></STRING></MODEL>
</ELEMENTTYPE>

<H3>Network Addresses</H3>
<ELEMENTTYPE NAME="telephone">
<EXPLAIN><TITLE>Telephone Number</TITLE>
<P>A telephone number is a string of alphanumerics and punctuation that uniquely identifies a telephone service terminal, including extension number.</P></EXPLAIN>
<MODEL><STRING></STRING></MODEL>
</ELEMENTTYPE>

<H3>Fax</H3>
<ELEMENTTYPE NAME="fax">
<EXPLAIN><TITLE>Fax Number</TITLE>
<P>A fax number is a string of alphanumerics and punctuation that uniquely identifies a fax service terminal.</P>
</EXPLAIN>
<MODEL><STRING></STRING></MODEL>

```
</ELEMENTTYPE>

<H3>Email</H3>
<ELEMENTTYPE NAME="email">
<EXPLAIN><TITLE>Email Address</TITLE>
<P>An email address is a datatype-constrained string that uniquely identifies a mailbox on a
server.</P></EXPLAIN>
<MODEL><STRING DATATYPE="email"></STRING></MODEL>
</ELEMENTTYPE>

<H3>Internet Address</H3>
<ELEMENTTYPE NAME="internet">
<EXPLAIN><TITLE>Internet Address</TITLE>
<P>An Internet address is a datatype-constrained string that uniquely identifies a resource on the Internet
by means of a URL.</P></EXPLAIN>
<MODEL><STRING DATATYPE="url"></STRING></MODEL>
</ELEMENTTYPE>

</DTD>
</SCHEMA>
```

**Service Description Sample**

```
<!DOCTYPE schema SYSTEM "bidl.dtd">
<SCHEMA>
<H1>Service Description Sample BID</H1>
<META
WHO.OWNS="Veo Systems"      WHO.COPYRIGHT="Veo Systems"
WHEN.COPYRIGHT="1998"       DESCRIPTION="Sample BID"
WHO.CREATED="*"             WHEN.CREATED="*"
WHAT.VERSION="*"            WHO.MODIFIED="*"
WHEN.MODIFIED="*"           WHEN.EFFECTIVE="*"
WHEN.EXPIRES="*"            WHO.EFFECTIVE="*"
WHO.EXPIRES="*">
</META>

<PROLOG>
<XMLDECL STANDALONE="no"></XMLDECL>
```

```
<DOCTYPE NAME="service">
<SYSTEM>service.dtd</SYSTEM></DOCTYPE>
</PROLOG>

<DTD NAME="service.dtd">
<H2>Services</H2>

<H3>Includes</H3>
<!-- INCLUDE><SYSTEM>comments.bim</SYSTEM></INCLUDE -->

<H3>Service Set</H3>
<ELEMENTTYPE NAME="service.set">
<EXPLAIN><TITLE>Service Set</TITLE>
<SYNOPSIS>A set of services.</SYNOPSIS></EXPLAIN>
<MODEL>
<ELEMENT NAME="service" OCCURS="+"></ELEMENT>
</MODEL></ELEMENTTYPE>

<H3>Services Prototype</H3>
<PROTOTYPE NAME="prototype.service">
<EXPLAIN><TITLE>Service</TITLE></EXPLAIN>
<MODEL><SEQUENCE>
<ELEMENT NAME="service.name"></ELEMENT>
<ELEMENT NAME="service.terms" OCCURS="+"></ELEMENT>
<ELEMENT NAME="service.location" OCCURS="+"></ELEMENT>
<ELEMENT NAME="service.operation" OCCURS="+"></ELEMENT>
</SEQUENCE></MODEL>
<!-- ATTGROUP><IMPLEMENTS HREF="common.attrib"></IMPLEMENTS></ATTGROUP -->
</PROTOTYPE>

<H3>Service</H3>
<INTRO><P>A service is an addressable network resource that provides interfaces to specific operations
by way of input and output documents.</P></INTRO>
<ELEMENTTYPE NAME="service">
<EXPLAIN><TITLE>Service</TITLE>
<P>A service is defined in terms of its name, the location(s) at which the service is available, and the
operation(s) that the service performs.</P></EXPLAIN>
<MODEL><SEQUENCE>
```

```
<ELEMENT NAME="service.name"></ELEMENT>
<ELEMENT NAME="service.location"></ELEMENT>
<ELEMENT NAME="service.operation" OCCURS="+"></ELEMENT>
<ELEMENT NAME="service.terms"></ELEMENT>
</SEQUENCE></MODEL>
</ELEMENTTYPE>

<H3>Service Name</H3>
<ELEMENTTYPE NAME="service.name">
<EXPLAIN><TITLE>Service Name</TITLE>
<P>The service name is a human-readable string that ascribes a moniker for a service. It may be
employed is user interfaces and documentation, or for other purposes.</P></EXPLAIN>
<MODEL><STRING></STRING></MODEL>
</ELEMENTTYPE>

<H3>Service Location</H3>
<ELEMENTTYPE NAME="service.location">
<EXPLAIN><TITLE>Service Location</TITLE>
<SYNOPSIS>A URI of a service.</SYNOPSIS>
<P>A service location is a datatype-constrained string that locates a service on the Internet by means of a
URI.</P></EXPLAIN>
<MODEL><STRING DATATYPE="url"></STRING></MODEL>
</ELEMENTTYPE>

<H3>Service Operations</H3>
<INTRO><P>A service operation consists of a name, location and its interface, as identified by the type
of input document that the service operation accepts and by the type of document that it will return as a
result.</P></INTRO>
<ELEMENTTYPE NAME="service.operation">
<EXPLAIN><TITLE>Service Operations</TITLE>
<P>A service operation must have a name, a location, and at least one document type as an input, with
one or more possible document types returned as a result of the operation.</P>
</EXPLAIN>
<MODEL><SEQUENCE>
<ELEMENT NAME="service.operation.name"></ELEMENT>
<ELEMENT NAME="service.operation.location"></ELEMENT>
<ELEMENT NAME="service.operation.input"></ELEMENT>
<ELEMENT NAME="service.operation.output"></ELEMENT>
```

```
</SEQUENCE></MODEL>
</ELEMENTTYPE>

<H3>Service Operation Name</H3>
<ELEMENTTYPE NAME="service.operation.name">
<EXPLAIN><TITLE>Service Operation Name</TITLE>
<P>The service operation name is a human-readable string that ascribes a moniker to a service operation. It may be employed in user interfaces and documentation, or for other purposes.</P></EXPLAIN>
<MODEL><STRING></STRING></MODEL>
</ELEMENTTYPE>

<H3>Service Operation Location</H3>
<INTRO><P>The service location is a network resource. That is to say, a URI.</P></INTRO>
<ELEMENTTYPE NAME="service.operation.location">
<EXPLAIN><TITLE>Service Operation Location</TITLE>
<SYNOPSIS>A URI of a service operation.</SYNOPSIS>
<P>A service operation location is a datatype-constrained string that locates a service operation on the Internet by means of a URL.</P></EXPLAIN>
<MODEL><STRING DATATYPE="url"></STRING></MODEL>
</ELEMENTTYPE>

<H3>Service Operation Input Document</H3>
<INTRO><P>The input to a service operation is defined by its input document type. That is, the service operation is invoked when the service operation location receives an input document whose type corresponds to the document type specified by this element.</P>
<P>Rather than define the expected input and output document types in the market participant document, this example provides pointers to externally-defined BIDs. This allows reuse of the same BID as the input and/or output document type for multiple operations. In addition, it encourages parallel design and implementation.</P></INTRO>
<ELEMENTTYPE NAME="service.operation.input">
<EXPLAIN><TITLE>Service Operation Input</TITLE>
<SYNOPSIS>Identifies the type of the service operation input document.</SYNOPSIS>
<P>Service location input is a datatype-constrained string that identifies a BID on the Internet by means of a URI.</P>
</EXPLAIN>
<MODEL><STRING DATATYPE="url"></STRING></MODEL>
</ELEMENTTYPE>
```

```
<H3>Service Operation Output Document Type</H3>
<INTRO><P>The output of a service operation is defined by its output document type(s). That is, the service operation is expected to emit a document whose type corresponds to the document type specified by this element.</P></INTRO>
<ELEMENTTYPE NAME="service.operation.output">
<EXPLAIN><TITLE>Service Operation Output</TITLE>
<SYNOPSIS>Identifies the type of the service operation output document.</SYNOPSIS>
<P>Service location output is a datatype-constrained string that identifies a BID on the Internet by means of a URI.</P>
</EXPLAIN>
<MODEL><STRING DATATYPE="url"></STRING></MODEL>
</ELEMENTTYPE>

<H3>Service Terms</H3>
<INTRO><P>This is a simple collection of string elements, describing the terms of an agreement.</P></INTRO>
<ELEMENTTYPE NAME="service.terms">
<EXPLAIN><TITLE>Service Terms</TITLE>
<SYNOPSIS>Describes the terms of a given agreement.</SYNOPSIS>
</EXPLAIN>
<MODEL><STRING DATATYPE="string"></STRING></MODEL>
</ELEMENTTYPE>

</DTD>
</SCHEMA>
```

The service DTD schema may be extended with a service type element in a common business language repository as follows:

```
<!ELEMENT service.type EMPTY>
<!ATTLIST service.type
        service.type.name (
        catalog.operator
        | commercial.directory.operator
        | eft.services.provider
        | escrower
        | fulfillment.service
        | insurer
        | manufacturer
        | market.operator
        | order.originator
        | ordering.service
```

```
        | personal.services.provider
        | retailer
        | retail.aggregator
        | schema.resolution.service
        | service.provider
        | shipment.acceptor
        | shipper
        | van
        | wholesale.aggregator
        ) #REQUIRED
        %common.attrib;
>
```

The service type element above illustrates interpretation information carried by a business interface definition, in this example a content form allowing identification of any one of a list of valid service types. Other interpretation information includes data typing, such as for example the element <H3>Internet Address</H3> including the content form "url" and expressed in the data type "string." Yet other interpretation information includes mapping of codes to elements of a list, such as for example the element <H3>State</H3> including the code mapping for states in the file "COUNTRY.US.SUBENTITY."

The service description referred to by the market participant DTD defines the documents that the service accepts and generates upon competition of the service. A basic service description is specified below as a XML document transact.dtd.

Transact.dtd models a transaction description, such as an invoice, or a description of an exchange of value. This document type supports many uses, so the transaction description element has an attribute that allows user to distinguish among invoices, performance, offers to sell, requests for quotes and so on. The exchange may occur among more than two parties, but only two are called out, the offeror and the counter party, each of whom is represented by a pointer to a document conforming to the market participant DTD outlined above. The counter party pointer is optional, to accommodate offers to sell. The exchange description is described in the module tranprim.mod listed below, and includes pricing and subtotals. Following the exchange description, charges applying to the transaction as a whole may be provided, and a total charge must be supplied. Thus, the transaction description schema document transact.dtd for this example is set forth below:

```
<!-- transact.dtd Version: 1.0 -->
<!-- Copyright 1998 Veo Systems, Inc. -->
```

```
...

<!ELEMENT transaction.description (meta?, issuer.pointer,
        counterparty.pointer?, exhange.descrption+, general.charges?,
        net.total?)>
<!ATTLIST transaction.description
        transaction.type (invoice | pro.forma | offer.to.sell | order
                | request.for.quote | request.for.bid
                | request.for.proposal | response.to.request.for.quote
                | response.to.request.for.bid
                | response.to.request.for.proposal) "invoice"
        %common.attrib;
        %altrep.attrib;
        %ttl.attrib;
>
```

Representative market participant, and service DTDs, created according to the definitions above, are as follows:

Market Participant DTD

```
<!ELEMENT business  (party.name+ , address.set) >
<!ATTLIST business business.number   CDATA   #REQUIRED
                >
<!ELEMENT party.name (#PCDATA )>
<!ELEMENT city (#PCDATA )>
<!ELEMENT internet (#PCDATA )>
<!ELEMENT country (#PCDATA )>
<!ELEMENT state (#PCDATA )>
<!ELEMENT email (#PCDATA )>
<!ELEMENT address.physical  (street , city , state , postcode? , country) >
<!ELEMENT telephone (#PCDATA )>
<!ELEMENT person  (party.name+ , address.set) >
<!ATTLIST person SSN   CDATA   #IMPLIED
                >
<!ELEMENT fax (#PCDATA )>
<!ELEMENT street (#PCDATA )>
<!ELEMENT address.set  (address.physical , telephone* , fax* , email* , internet*) >
<!ELEMENT postcode (#PCDATA )>
<!ELEMENT market.participant  (business | person) >
```

Service DTD

```
<!ELEMENT service.location (#PCDATA )>
<!ELEMENT service.terms (#PCDATA )>
<!ELEMENT service.operation.name (#PCDATA )>
<!ELEMENT service.operation  (service.operation.name , service.operation.location ,
service.operation.input , service.operation.output) >
<!ELEMENT service  (service.name , service.location , service.operation+ , service.terms) >
<!ELEMENT service.operation.input (#PCDATA )>
<!ELEMENT service.operation.location (#PCDATA )>
<!ELEMENT service.name (#PCDATA )>
<!ELEMENT service.set (service+ )>
<!ELEMENT service.operation.output (#PCDATA )>
```

One instance of a document produced according to the transact.dtd follows:

```
<?xml version="1.0"?>
<!-- rorder.xml Version:  1.0 -->
<!-- Copyright 1998 Veo Systems, Inc. -->

<!DOCTYPE transaction.description SYSTEM "urn:x-veosystems:dtd:cbl:transact:1.0:>
<transaction.description transaction.type="order">
<meta>
 <urn?urn:x-veosystems:doc:00023
 </urn>
        <thread.id party.assigned.by="reqorg">FRT876
        </thread.id>
</meta>
        <issuer.pointer>
                <xll.locator urllink="reqorg.xml">Customer
                Pointer
                </xll.locator>
        </issuer.pointer>
        <counterparty.pointer>
                <xll.locator urllink="compu.xml">Catalog entry owner        pointer
                </xll.locator>
        </counterparty.pointer>
<exchange.description>
<line.item>
        <product.instance>
        <product.description.pointer>
                <xll.locator urllink="cthink.xml">Catalogue Entry    Pointer
                </xll.locator>
        </product.description.pointer>
        <product.specifics>
        <info.description.set>
<info.description>
 <xml.descriptor>
        <doctype>
        <dtd system.id="urn:x-veosystems:dtd:cbl:gprod:1.0"/>
        </doctype>
        <xml.descriptor.details>
        <xll.xptr.frag>DESCENDANT(ALL,os)STRING("Windows                95")
        </xll.xptr.frag>
        <xll.xptr.frag>DECENDANT(ALL,p.speed)STRING("200")
        </xll.xptr.frag>
        <xll.xptr.frag>DESCENDANT(ALL,hard.disk.capacity)
                STRING("4")
        </xll.xptr.frag>
        <xll.xptr.frag>DESCENDANT(ALL,d.size)STRING("14.1")
        </xll.xptr.frag>
        </xml.descriptor.details>
 </xml.descriptor>
</info.description>
        </info.description.set>
        </product.specifics>
<quantity>1
</quantity>
        </product.instance>
<shipment.coordinates.set>
```

```
<shipment.coordinates>
<shipment.destination>
        <address.set>
    <address.named>SW-1
    </address.named>
    <address.physical>
        <building.sublocation>208C</building.sublocation>
        <location.in.street>123
        </location.in.street>
        <street>Frontage Rd.
        </street>
        <city>Beltway
        </city>
        <country.subentity.us
        country.subentity.us.name="MD"/>
        <postcode>20000
        </postcode>
    </address.physical>
    <telephone>
        <telephone.number>617-666-2000
        </telephone.number>
        <telephone.extension>1201
        </telephone.extension>
    </telephone>
    </address.set>
</shipment.destination>

<shipment.special>No deliveries after 4 PM</shipment.special>
</shipment.coordinates>
</shipment.coordinates.set>
    <payment.set>
    <credit.card
    issuer.name="VISA"
    instrument.number="3787-812345-67893"
    expiry.date="12/97"
    currency.code="USD"/>
    <amount.group>
        <amount.monetary currency.code="USD">3975
        </amount.monetary>
    </amount.group>
    </payment.set>
</line.item>
</exhange.description>
</transaction.description>
```

Accordingly, the present invention provides a technique by which a market participant is able to identify itself, and identify the types of input documents and the types of output documents with which it is willing to transact business. The particular manner in which the content carried in such documents is processed by the other parties to the transaction, or by the local party, is not involved in establishing a business relationship nor carrying out transactions.

Fig. 9 provides a simplified view of a participant node in a network practicing aspects of the present invention. The node illustrated in Fig. 9 includes a network interface 900 which is coupled to a communication network on port 901. The network interface is coupled to a document parser 901. The parser 901 supplies the logical structures from an incoming document to a translator module 902, which provides for translating the incoming document into a form usable by the host transaction system, and vice versa translating the output of host processes into the format of a document which matches the output document form in the business interface definition for transmission to a destination. The parser 901 and translator 902 are responsive to the business interface definition stored in the participant module 903.

The output data structures from the translator 902 are supplied to a transaction process front end 904 along with events signaled by the parser 901. The front end 904 in one embodiment consists of a JAVA virtual machine or other similar interface adapted for communication amongst diverse nodes in a network. The transaction processing front end 904 responds to the events indicated by the parser 901 and the translator 902 to route the incoming data to appropriate functions in the enterprise systems and networks to which the participant is coupled. Thus, the transaction process front end 904 in the example of Fig. 9 is coupled to commercial functions 905, database functions 906, other enterprise functions such as accounting and billing 907, and to the specific event listeners and processors 908 which are designed to respond to the events indicated by the parser.

The parser 901 takes a purchase order like that in the example above, or other document, specified according to the business interface definition and creates a set of events that are recognized by the local transaction processing architecture, such as a set of JAVA events for a JAVA virtual machine.

The parser of the present invention is uncoupled from the programs that listen for events based on the received documents. Various pieces of mark-up in a received document or a complete document meeting certain specifications serve as instructions for listening functions to start processing. Thus listening programs carry out the business logic associated with the document information. For example, a program associated with an address element may be code that validates the postal code by checking the database. These listeners subscribe to events by registering with a

document router, which directs the relevant events to all subscribers who are interested in them.

For example, the purchase order specified above may be monitored by programs listening for events generated by the parser, which would connect the document or its contents to an order entry program. Receipt of product descriptions within the purchase order, might invoke a program to check inventory. Receipt of address information within the purchase order, would then invoke a program to check availability of services for delivery. Buyer information fields in the document, could invoke processes to check order history for credit worthiness or to offer a promotion or similar processing based on knowing the identity of the consumer.

Complex listeners can be created as configurations of primitive ones. For example, a purchase order listener may contain and invoke the list listeners set out in the previous paragraph, or the list members may be invoked on their own. Note that the applications that the listeners run are unlikely to be native XML processes or native JAVA processes. In these cases, the objects would be transformed into the format required by the receiving trans application. When the application finishes processing, its output is then transformed back to the XML format for communication to other nodes in the network.

It can be seen that the market participant document type description, and the transaction document type description outlined above include a schematic mapping for logic elements in the documents, and include mark-up language based on natural language. The natural language mark-up, and other natural language attributes of XML facilitate the use of XML type mark-up languages for the specification of business interface definitions, service descriptions, and the descriptions of input and output documents.

The participant module 903 in addition to storing the business interface definition includes a compiler which is used to compile objects or other data structures to be used by the transaction process front end 904 which corresponds to the logical structures in the incoming documents, and to compile the translator 902. Thus, as the business interface definition is modified or updated by the participant as the transactions with which the participant is involved change, the translator 902 and parser 901 are automatically kept up to date.

In one system practicing aspects of the present invention, the set of JAVA events is created by a compiler which corresponds to the grove model of SGML, mainly the standard Element Structure Information Set augmented by the "property set" for each element. *International Standard ISO/IEC 10179:1996 (E), Information Technology -- Processing Languages -- Document Style Semantics and Specification Language (DSSSL).* Turning the XML document into a set of events for the world to process contrasts with the normal model of parsing in which the parser output is maintained as an internal data structure. By translating the elements of the XML document into JAVA events or other programming structures that are suitable for use by the transaction processing front end of the respective nodes enables rich functionality at nodes utilizing the documents being traded.

Thus, the transaction process front end 904 is able to operate in a publish and subscribe architecture that enables the addition of new listener programs without the knowledge of or impact on other listening programs in the system. Each listener, 905, 906, 907, 908 in Fig. 9, maintains a queue in which the front end 904 directs events. This enables multiple listeners to handle events in parallel at their own pace.

Furthermore, according to the present invention the applications that the listeners run need not be native XML functions, or native functions which match the format of the incoming document. Rather, these listeners may be JAVA functions, if the transaction process front end 904 is a JAVA interface, or may be functions which run according to a unique transaction processing architecture. In these cases, the objects would be transformed into the format required by the receiving application. When the application of the listener finishes, its output is then transformed back into the format of a document as specified by the business interface definition in the module 903. Thus, the translator 902 is coupled to the network interface 900 directly for supplying the composed documents as outputs.

The listeners coupled to the transaction processing front end may include listeners for input documents, listeners for specific elements of the input documents, and listeners for attributes stored in particular elements of the input document. This enables diverse and flexible implementations of transaction processes at the participant nodes for filtering and responding to incoming documents.

Fig. 10 illustrates a process of receiving and processing an incoming document for the system of Fig. 9. Thus, the process begins by receiving a document at the

network interface (step 1000). The parser identifies the document type (1001) in response to the business interface definition. Using the business interface definition, which stores a DTD for the document in the XML format, the document is parsed (step 1002). Next, the elements and attributes of the document are translated into the format of the host (step 1003). In this example, the XML logic structures are translated into JAVA objects which carry the data of the XML element as well as methods associated with the data such as get and set functions. Next, the host objects are transferred to the host transaction processing front end (step 1004). These objects are routed to processes in response to the events indicated by the parser and the translator. The processes which receive the elements of the document are executed and produce an output (step 1005). The output is translated to the format of an output document as defined by the business interface definition (step 1006). In this example, the translation proceeds from the form of a JAVA object to that of an XML document. Finally, the output document is transmitted to its destination through the network interface (step 1007).

Fig. 11 is a more detailed diagram of the event generator/event listener mechanism for the system of Fig. 9. In general the approach illustrated in Fig. 11 is a refinement of the JAVA JDK 1.1 event model. In this model, three kinds of objects are considered. A first kind of object is an event object which contains information about the occurrence of an event. There may be any number of kinds of event objects, corresponding to all the different kinds of events which can occur. A second kind of object is an Event generator, which monitors activity and generates event objects when something happens. Third, event listeners, listen for event objects generated by event generators. Event listeners generally listen to specific event generators, such as for mouse clicks on a particular window. Event listeners call an "ADD event listener" method on the event generator. This model can be adapted to the environment of Fig. 9 in which the objects are generated in response to parsing and walking a graph of objects, such as represented by an XML document.

The system illustrated in Fig. 11 includes a generic XML parser 1100. Such parser can be implemented using a standard call back model. When a parsing event occurs, the parser calls a particular method in an application object, passing in the appropriate information in the parameters. Thus a single application 1101 resides with the parser. The application packages the information provided by the parser in an XML event object and sends it to as many event listeners as have identified themselves, as

indicated by the block 1102. The set of events 1102 is completely parser independent. The events 1102 can be supplied to any number of listeners and any number of threads on any number of machines. The events are based on the element structure information set ESIS in one alternative. Thus, they consist of a list of the important aspects of a document structure, such as the starts and ends of elements, or of the recognition of an attribute. XML (and SGML) parsers generally use the ESIS structure as a default set of information for a parser to return to its application.

A specialized ESIS listener 1103 is coupled to the set of events 1102. This listener 1103 implements the ESIS listener API, and listens for all XML events from one or more generators. An element event generator 1104 is a specialized ESIS listener which is also an XML event generator. Its listeners are objects only interested in events for particular types of elements. For example in an HTML environment, the listener may only be interested in ordered lists, that is only the part of the document between the <OL> and </OL>tags. For another example, a listener may listen for "party.name" elements, or for "service.name" elements according to the common business language, from the example documents above, process the events to ensure that the elements carry data that matches the schematic mapping for the element, and react according to the process needed at the receiving node.

This allows the system to have small objects that listen for particular parts of the document, such as one which only adds up prices. Since listeners can both add and remove themselves from a generator, there can be a listener which only listens to for example the <HEAD> part of an HTML document. Because of this and because of the highly recursive nature of XML documents, it is possible to write highly targeted code, and to write concurrent listeners. For example, an <OL> listener can set up an <LI> listener completely separate from the manner in which the <UL> (unordered list) listener sets up its <LI> listener. Alternatively, it can create a listener which generates a graphic user interface and another which searches a database using the same input. Thus, the document is treated as a program executed by the listeners, as opposed to the finished data structure which the application examines one piece at a time. If an application is written this way, it is not necessary to have the entire document in memory to execute an application.

The next listener coupled to the set of events 1102 is an attribute filter 1105. The attribute filter 1105 like the element filter 1104 is an attribute event generator according

to the ESIS listener model. The listener for an attribute filter specifies the attributes it is interested in, and receives events for any element having that attribute specified. So for example, a font manager might receive events only for elements having a font attribute, such as the <P FONT= "Times Roman" /P>.

The element event generator 1104 supplies such element objects to specialize the element listeners 1104A.

The attribute event generator 1105 supplies the attribute event objects to attribute listeners 1105A. Similarly, the attribute objects are supplied to a "architecture" in the sense of an SGML/XML transformation from one document type to another using attributes. Thus the architecture of 1105B allows a particular attribute with a particular name to be distinguished. Only elements with that attribute defined become part of the output document, and the name of the element in the output document is the value of the attribute in the input document. For example, if the architecture 1105B is HTML, the string: <PURCHASES HTML="OL"><ITEM HTML="LI"><NAME HTML="B">STUFF</NAME><PRICE HTML="B">123</PRICE></ITEM></PURCHASES>

translates into:

<OL><LI><B>STUFF</B><B>123</B></LI></OL>

which is correct HTML.

The next module which is coupled to the set of events 1102 is a tree builder 1106. The tree builder takes a stream of XML events and generates a tree representation of the underlying document. One preferred version of the tree builder 1106 generates a document object model DOM object 1107, according to the specification of the W3C (See, http://www.w3.org/TR/1998/ WD-DOM-19980720/ introduction.html). However listeners in event streams can be used to handle most requirements, a tree version is useful for supporting queries around a document, reordering of nodes, creation of new documents, and supporting a data structure in memory from which the same event stream can be generated multiple times, for example like parsing the document many times. A specialized builder 1108 can be coupled to the tree builder 1106 in order to build special subtrees for parts of the document as suits a particular implementation.

In addition to responses to incoming documents, other sources of XML events 1102 can be provided. Thus, an event stream 1110 is generated by walking over a tree of DOM objects and regenerating the original event stream created when the document was

being parsed. This allows the system to present the appearance that the document is being parsed several times.

The idea of an object which walks a tree and generates a stream of events can be generalize beyond the tree of DOM objects, to any tree of objects which can be queried. Thus, a JAVA walker 1112 may be an application which walks a tree of JAVA bean components 1113. The walker walks over all the publicly accessible fields and methods. The walker keeps track of the objects it has already visited to ensure that it doesn't go into an endless cycle. JAVA events 1114 are the type of events generated by the JAVA walker 1112. This currently includes most of the kinds of information one can derive from an object. This is the JAVA equivalent of ESIS and allows the same programming approach applied to XML to be applied to JAVA objects generally, although particularly to JAVA beans.

The JAVA to XML event generator 1115 constitutes a JAVA listener and a JAVA event generator. It receives the stream of events 1114 from the JAVA walker 1112 and translates selected ones to present a JAVA object as an XML document. In the one preferred embodiment, the event generator 1115 exploits the JAVA beans API. Each object seen becomes an element, with the element name the same as the class name. Within that element, each embedded method also becomes an element whose content is the value returned by invoking the method. If it is an object or an array of objects, then these are walked in turn.

Fig. 12 outlines a particular application built on the framework of Fig. 11. This application takes in an XML document 1200 and applies it to a parser/generator 1201. ESIS events 1202 are generated and supplied to an attribute generator 1203 and tree builder 1204. The attribute generator corresponds to the generator 505 of Fig. 5. It supplies the events to the "architecture" 505B for translating the XML input to an HTML output for example. These events are processed in parallel as indicated by block 1205 and processed by listeners. The output of the listeners are supplied to a document writer 506 and then translated back to an XML format for output. Thus for example this application illustrated in Fig. 12 takes an XML document and outputs an HTML document containing a form. The form is then sent to a browser, and the result is converted back to XML. For this exercise, the architecture concept provides the mapping from XML to HTML. The three architectures included in Fig. 12 include one for providing the structure of the HTML document, such as tables and lists, a second

specifying text to be displayed, such as labels for input fields on the browser document, and the third describes the input fields themselves. The elements of the XML document required to maintain the XML documents structure become invisible fields in the HTML form. This is useful for use in reconstruction of the XML document from the information the client will put into the HTTP post message that is sent back to the server. Each architecture takes the input document and transforms it into an architecture based on a subset of HTML. Listeners listening for these events, output events for the HTML document, which then go to a document writer object. The document writer object listens to XML events and turns them back into an XML document. The document writer object is a listener to all the element generators listening to the architectures in this example.

The organization of the processing module illustrated in Figs. 11 and 12 is representative of one embodiment of the parser and transaction process front end for the system of Fig. 9. As can be seen, a very flexible interface is provided by which diverse transaction processes can be executed in response to the incoming XML documents, or other structured document formats.

Fig. 13 illustrates a node similar to that of Fig. 9, except that it includes a business interface definition builder module 1300. Thus, the system of Fig. 13 includes a network interface 1301, a document parser 1302, and a document translator 1303. The translator 1303 supplies its output to a transaction processing front end 1304, which in turn is coupled to listening functions such as commercial functions 1305, a database 1306, enterprise functions 1307, and other generic listeners and processors 1308. As illustrated in Fig. 13, the business interface definition builder 1300 includes a user interface, a common business library CBL repository, a process for reading complementary business interface definitions, and a compiler. The user interface is used to assist an enterprise in the building of a business interface definition relying on the common business library repository, and the ability to read complementary business interface definitions. Thus, the input document of a complementary business interface definition can be specified as the output document of a particular transaction, and the output document of the complementary business interface definition can be specified as the input to such transaction process. In a similar manner a transaction business interface definition can be composed using components selected from the CBL repository. The use of the CBL repository encourages the use of standardized document

formats, such as the example schema (bid1) documents above, logical structures and interpretation information in the building of business interface definitions which can be readily adopted by other people in the network.

The business interface definition builder module 1300 also includes a compiler which is used for generating the translator 1303, the objects to be produced by the translator according to the host transaction processing architecture, and to manage the parsing function 1302.

Fig. 14 is a heuristic diagram showing logical structures stored in the repository in the business interface definition builder 1300. Thus, the repository storage representative party business interface definitions 1400, including for example a consumer BID 1401, a catalog house BID 1402, a warehouse BID 1403, and an auction house BID 1404. Thus, a new participant in an online market may select as a basic interface description one of the standardized BIDs which best matches its business. In addition, the repository will store a set of service business interface definitions 1405. For example, an order entry BID 1406, an order tracking BID 1407, an order fulfillment BID 1408, and a catalog service BID 1409 could be stored. As a new participant in the market builds a business interface definition, it may select the business interface definitions of standardized services stored in the repository.

In addition to the party and service BIDs, input and output document BIDs are stored in the repository as indicated by the field 1410. Thus, a purchase order BID 1411, an invoice BID 1412, a request for quote BID 1413, a product availability report BID 1414, and an order status BID 1415 might be stored in the repository.

The repository, in addition to the business interface definitions which in a preferred system are specified as document type definitions according to XML, stores interpretation information in the form of semantic maps as indicated by the field 1416. Thus, semantic maps which are used for specifying weights 1417, currencies 1418, sizes 1419, product identifiers 1420, and product features 1421 in this example might be stored in the repository. Further, the interpretation information provides for typing of data structures within the logical structures of documents.

In addition, logical structures used in the composing of business interface definitions could be stored in the repository as indicated by block 1422. Thus, forms for providing address information 1423, forms for providing pricing information 1424, and forms for providing terms of contractual relationships could be provided 1425. As the

network expands, the CBL repository will also expand and standardize tending to make the addition of new participants, and the modification of business interface definitions easier.

Fig. 15 illustrates the process of building a business interface definition using the system of Fig. 13. The process begins by displaying a BID builder graphical interface to the user (step 1500). The system accepts user input identifying a participant, service and document information generated by the graphical interface (step 1501).

Next, any referenced logical structures, interpretation information, document definitions and/or service definitions are retrieved from the repository in response to user input via the graphical user interface (step 1502). In the next step, any complementary business interface definitions or components of business interface definitions are accessed from other participants in the network selected via user input, by customized search engines, web browsers or otherwise (step 1503). A document definition for the participant is created using the information gathered (step 1504). The translators for the document to host and host to document mappers are created by the compiler (step 1505). Host architecture data structures corresponding to the definition are created by the compiler (step 1506), and the business interface definition which has been created is posted on the network, such as by posting on a website or otherwise, making it accessible to other nodes in the network (step 1507).

Business interface definitions tell potential trading partners the online services the company offers and which documents to use to invoke those services. Thus, the services are defined in the business interface definition by the documents that they accept and produce. This is illustrated in the following fragment of an XML service definition.

```
<service>
<service.name>Order Service</service.name>
<service.location>www.veosystems.com/order</service.location>
<service.op>
<service.op.name>Submit Order</service.op.name>
<service.op.inputdoc>www.commerce.net/po.dtd</service.op.inputdoc>
<service.op.outputdoc>
        www.veosystems.com/invoice.dtd</service.op.outputdoc>
</service.op>
< service.op>
```

```
< service.op.name>Track Order</service.op.name>
<service.op.inputdoc> www.commerce.net
        /request.track.dtd<service.op.inputdoc>
<service.op.outputdoc>
        www.veosystems.com/response.track.dtd<service.op.outputdoc>
</service.op>
</service>
```

This XML fragment defines a service consisting of two transactions, one for taking orders and the other for tracking them. Each definition expresses a contract or promise to carry out a service if a valid request is submitted to the specified Web address. The Order service here requires an input document that conforms to a standard "po.dtd" Document Type Definition located in the repository, which may be local, or stored in an industry-wide registry on the network. If a node can fulfill the order, it will return a document conforming to a customized "invoice.dtd" whose definition is local. In effect, the company is promising to do business with anyone who can submit a Purchase Order that conforms to the XML specification it declares. No prior arrangement is necessary.

The DTD is the formal specification or grammar for documents of a given type; it describes the elements, their attributes, and the order in which they must appear. For example, purchase orders typically contain the names and addresses of the buyer and seller, a set of product descriptions, and associated terms and conditions such as price and delivery dates. In Electronic Data Interchange EDI for example, the X12 850 specification is a commonly used model for purchase orders.

The repository encourages the development of XML document models from reusable semantic components that are common to many business domains. Such documents can be understood from their common message elements, even though they may appear quite different. This is the role of the Common Business Library repository.

The Common Business Library repository consists of information models for generic business concepts including:

- business description primitives like companies, services, and products;
- business forms like catalogs, purchase orders, and invoices;
- standard measurements, date and time, location, classification codes.

This information is represented as an extensible, public set of XML building blocks that companies can customize and assemble to develop XML applications quickly. Atomic CBL elements implement industry messaging standards and conventions such as standard ISO codes for countries, currencies, addresses, and time. Low level CBL semantics have also come from analysis of proposed metadata frameworks for Internet resources, such as Dublin Core.

The next level of elements use these building blocks to implement the basic business forms such as those used in X12 EDI transactions as well as those used in emerging Internet standards such as OTP (Open Trading Protocol) and OBI (Open Buying on the Internet).

CBL's focus is on the functions and information that are common to all business domains (business description primitives like companies, services, and products; business forms like catalogs, purchase orders, and invoices; standard measurements, date and time, location, classification codes). CBL builds on standards or industry conventions for semantics where possible (e.g., the rules that specify "day/month/year" in Europe vs "month/day/year" in the U.S. are encoded in separate CBL modules).

The CBL is a language that is used for designing applications. It is designed to bridge the gap between the "document world" of XML and the "programming world" of JAVA or other transaction processing architectures. Schema embodies a philosophy of "programming with documents" in which a detailed formal specification of a document type is the master source from which a variety of related forms can be generated. These forms include XML DTDs for CBL, JAVA objects, programs for converting XML instances to and from the corresponding JAVA objects, and supporting documentation.

The CBL creates a single source from which almost all of the pieces of a system can be automatically generated by a compiler. The CBL works by extending SGML/XML, which is normally used to formally define the structures of particular document types, to include specification of the semantics associated with each information element and attribute. The limited set of (mostly) character types in SGML/XML can be extended to declare any kind of datatype.

Here is a fragment from the CBL definition for the "datetime" module:

```
<!-- datetime.mod  Version:  1.0 -->
<!-- Copyright 1998 Veo Systems, Inc. -->
...
<! ELEMENT year (#PCDATA)>
<! ATTLIST year
```

```
        schema CDATA #FIXED "urn:x-veosystems:stds:iso:8601:3.8"
>

<! ELEMENT month (#PCDATA)>
<! ATTLIST month
        schema CDATA #FIXED "urn:x-veosystems:stds:iso:8601:3.12"
>

...
```

In this fragment, the ELEMENT "year" is defined as character data, and an associated "schema" attribute, also character data, defines the schema for "year" to be section 3.8 of the ISO 8601 standard.

This "datetime" CBL module is in fact defined as an instance of the Schema DTD. First, the module name is defined. Then the "datetime" element "YEAR" is bound to the semantics of ISO 8601:

```
<! DOCTYPE SCHEMA SYSTEM "schema.dtd">
<SCHEMA><H1>Date and Time Module</H1>

...
<ELEMNTTYPE NAME="year" DATATYPE="YEAR"><MODEL>
        <STRING
DATATYPE="YEAR"></STRING></MODEL>
<ATTDEF NAME=:schema:iso8601" DATATYPE="CDATA">
        <FIXED>3.8
Gregorian calendar</FIXED></ATTDEF></ELEMENTTYPE>

...
```

The example market participant and service modules above are also stored in the CBL repository.

In Fig. 16, an Airbill 1600 is being defined by customizing a generic purchase order DTD 1601, adding more specific information about shipping weight 1602. The generic purchase order 1601 was initially assembled from the ground up out of CBL modules for address, date and time, currency, and vendor and product description. Using CBL thus significantly speeds the development and implementation of XML commerce applications. More importantly, CBL makes it easier for commercial applications to be interconnected.

In the CBL, XML is extended with a schema. The extensions add strong-typing to XML elements so that content can be readily validated. For example, an element called <CPU_clock_speed> can be defined as an integer with a set of valid values: {100, 133, 166, 200, 233, 266 Mhz.}. The schema also adds class-subclass hierarchies, so that

information can be readily instantiated from class definitions. A laptop, for instance, can be described as a computer with additional tags for features such as display type and battery life. These and other extensions facilitate data entry, as well as automated translations between XML and traditional Object-Oriented and relational data models.

Thus the completed BID is run through the compiler which produces the DTDs for the actual instance of a participant and a service as outlined above, the JAVA beans which correspond to the logical structures in the DTD instances, and transformation code for transforming from XML to JAVA and from JAVA to XML. In alternative systems documentation is also generated for display on a user interface or for printing by a user to facilitate use of the objects. An example of this is found in U.S. patent application No. 09/173,858, entitled DOCUMENTS FOR COMMERCE IN TRADING PARTNER NETWORKS AND INTERFACE DEFINITIONS BASED ON THE DOCUMENTS which is hereby incorporated by reference.

An application of CBL and the BID processor of the present invention in an XML/JAVA environment can be further understood by the following explanation of the processing of a Purchase Order.

Company A defines its Purchase Order document type using a visual programming environment that contains a library of CBL DTDs and modules, all defined using common business language elements so that they contain data type and other interpretation information. Company A's PO might just involve minor customizations to a more generic "transaction document" specification that comes with the CBL library, or it might be built from the ground up from CBL modules for address, date and time, currency, etc.

The documentation for the generic "transaction document" specification (such as the transact.dtd set out above) typifies the manner in which CBL specifications are built from modules and are interlinked with other CBL DTDs.

A compiler takes the purchase order definition and generates several different target forms. All of these target forms can be derived through "tree to tree" transformations of the original specification. The most important for this example are:

(a) the XML DTD for the purchase order.

(b) a JAVA Bean that encapsulates the data structures for a purchase order (the JAVA classes, arguments, datatypes, methods, and exception

structures are created that correspond to information in the Schema definition of the purchase order).

(c) A "marshaling" program that converts purchase orders that conform to the Purchase Order DTD into a Purchase Order JAVA Bean or loads them into a database, or creates HTML (or an XSL style sheet) for displaying purchase orders in a browser.

(d) An "unmarshaling" program that extracts the data values from Purchase Order JAVA Beans and converts them into an XML document that conforms to the Purchase Order DTD.

Now, back to the scenario. A purchasing application generates a Purchase Order that conforms to the DTD specified as the service interface for a supplier who accepts purchase orders.

The parser uses the purchase order DTD to decompose the purchase order instance into a stream of information about the elements and attribute values it contains. These "property sets" are then transformed into corresponding JAVA event objects by wrapping them with JAVA code. This transformation in effect treats the pieces of marked-up XML document as instructions in a custom programming language whose grammar is defined by the DTD. These JAVA events can now be processed by the marshaling applications generated by the compiler to "load" JAVA Bean data structures.

Turning the XML document into a set of events for JAVA applications to process, is unlike the normal model of parsing in which the parser output is maintained as an internal data structure and processing does not begin until parsing completes. The event based processing, in response to the BID definitions, is the key to enabling the much richer functionality of the processor because it allows concurrent document application processing to begin as soon as the first event is emitted.

JAVA programs that "listen for" events of various types are generated from the Schema definition of those events. These listeners are programs created to carry out the business logic associated with the XML definitions in the CBL; for example, associated with an "address" element may be code that validates the postal code by checking a

database. These listeners "subscribe" to events by registering with the document router, which directs the relevant events to all the subscribers who are interested in them.

This publish and subscribe architecture means that new listener programs can be added without knowledge by or impact on existing ones. Each listener has a queue into which the router directs its events, which enables multiple listeners can handle events in parallel at their own pace.

For the example purchase order here, there might be listeners for:

- the purchase order, which would connect it to an order entry program,
- product descriptions, which might check inventory,
- address information, which could check Fed Ex or other service for delivery availability,
- buyer information, which could check order history (for creditworthiness, or to offer a promotion, or similar processing based on knowing who the customer is).

Complex listeners can be created as configurations of primitive ones (e.g., a purchase order listener may contain and invoke these listeners here, or they may be invoked on their own).

Fig. 17 illustrates the market maker node in the network of Fig. 3. The market maker node includes the basic structures of the system of Fig. 9, including a network interface 1701, a document parser 1702, a document to host and host to document translator 1703, and a front end 1704, referred to as a router in this example. The market maker module 1705 in this example includes a set of business interface definitions, or other identifiers sufficient to support the market maker function, for participants in the market, a CBL repository, and a compiler all serving the participants in the market. The router 1704 includes a participant registry and document filters which respond to the events generated at the output of the translator and by the parser to route incoming documents according to the participant registry and according to the element and attribute filters amongst the listeners to the XML event generators. Thus, certain participants in the market may register to receive documents that meet prespecified parameters. For example, input documents according to a particular DTD, and including an attribute such as numbers of products to be purchased greater than a threshold, or such as a maximum price of a document request to be purchased, can be used to filter documents at the router 1704. Only such documents as match the information registered

in the participant registry at the router 1704 are then passed on to the registered participant.

The router 1704 may also serve local host services 1705 and 1706, and as such act as a participant in the market as well as the market maker. Typically, documents that are received by the router 1704 are traversed to determine the destinations to which such documents should be routed, there again passed back through the translator 1703, if necessary, and out the network interface 1701 to the respective destinations.

The market maker is a server that binds together a set of internal and external business services to create a virtual enterprise or trading community. The server parses incoming documents and invokes the appropriate services by, for example, handing off a request for product data to a catalog server or forwarding a purchase order to an ERP system. The server also handles translation tasks, mapping the information from a company's XML documents onto document formats used by trading partners and into data formats required by its legacy systems.

With respect to the service definition above, when a company submits a purchase order, the XML parser in the server uses the purchase order DTD to transform the purchase order instance into a stream of information events. These events are then routed to any application that is programmed to handle events of a given type; in some cases, the information is forwarded over the Internet to a different business entirely. In the purchase order example, several applications may act on information coming from the parser:

- An order entry program processes the purchase order as a complete message;
- An ERP system checks inventory for the products described in the purchase order;
- A customer database verifies or updates the customer's address;
- A shipping company uses the address information to schedule a delivery
- A bank uses the credit card information to authorize the transaction.

Trading partners need only agree on the structure, content, and sequencing of the business documents they exchange, not on the details of APIs. How a document is processed and what actions result is strictly up to the business providing the service. This elevates integration from the system level to the business level. It enables a business

to present a clean and stable interface to its business partners despite changes in its internal technology implementation, organization, or processes.

Figs. 18, 19 and 20 illustrate processes executed at a market maker node in the system of Fig. 17. In Fig. 18, an input document is received at the network interface from an originating participant node (step 1800). The document is parsed (step 1801). The document is translated to the format of the host, for example XML to JAVA (step 1802). The host formatted events and objects are then passed to the router service (step 1803). The services registered to accept the document according to the document type and content of the document are identified (step 1804). The document or a portion of the document is passed to the identified services (step 1805). As service is performed in response to the document content (step 1806). The output data of the service is produced (step 1807). The output is converted to the document format, for example from a JAVA format to an XML format (step 1808). Finally, the output document is sent to a participant node (step 1809).

The registration service is one such function which is managed by the router. Thus, a market participant document is accepted at the network interface as shown in Fig. 19 (step 1900). The market participant document is stored in the business interface definition repository (step 1901) for the market maker node. In addition, the document is parsed (step 1902). The parsed document is translated into the format of the host (step 1903). Next, the document is passed to the router service (step 1904). The router service includes a listener which identifies the registration service as the destination of the document according to the document type and content (step 1905). The document or elements of the document are passed to the registration service (step 1906). In the registration service, the needed service specifications are retrieved according to the business interface definition (step 1907). If the service specifications are gathered, at step 1908, the router service filters are set according to the business interface definition and the service specifications (step 1909). Registration acknowledgment data is produced (1910). The registration acknowledgment data is converted to a document format (step 1911). Finally, the acknowledgment document is sent to the participant node indicating to the participant that is successfully registered with the market maker (step 1912).

The process at step 1907 of gathering needed service specifications is illustrated for one example in Fig. 20. This process begins by locating a service business interface

definition supported by the market participant (step 2000). The service definition is retrieved, for example by an E-mail transaction or web access to repository node (step 2001). The service specification is stored in the BID repository (step 2002). The service business interface definition document is parsed (step 2003). The parsed document is translated into the format of the host (step 2004). Host objects are passed to the router service (step 2005). The registration service is identified according to the document type and content (step 2006). Finally, the information in the service business interface definition document is passed to the registration service (step 2007) for use according to the process of Fig. 19.

Fig. 21 illustrates the processor, components and sequence of processing of incoming data at market maker node according to the present invention. The market maker node includes a communication agent 2100 at the network interface. The communication agent is coupled with an XML parser 2101 which supplies events to an XML processor 2102. The XML processor supplies events to a document router. The document router feeds a document service 2104 that provides an interface for supplying the received documents to the enterprise solution software 2105 in the host system. The communication agent 2100 is an Internet interface which includes appropriate protocol stacks supporting such protocols as HTTP, SMTP, FTP, or other protocols. Thus, the incoming data could come in an XML syntax, an ASCII data syntax or other syntax as suits a particular communication channel. All the documents received in non-XML syntaxes are translated into XML and passed the XML parser. A translation table 2106 is used to support the translation from non-XML form into XML form.

The converted documents are supplied to the parser 2101. The XML parser parses the received XML document according to the document type definition which matches it. If an error is found, then the parser sends the document back to the communication agent 2100. A business interface definition compiler BIDC 2107 acts as a compiler for business interface definition data. The DTD file for the XML parser, JAVA beans corresponding to the DTD file, and translation rules for translating DTD files to JAVA beans are created by compiling the BID data. An XML instance is translated to JAVA instance by referring to these tools. Thus the BID compiler 2107 stores the DTD documents 2108 and produces JAVA documents which correspond 2109. The XML documents are passed to the processor 2102 which translates them into the JAVA format. In a preferred system, JAVA documents which have the same status

as the document type definitions received in the XML format are produced. The JAVA beans are passed to the document router 2103. The document router 2103 receives the JAVA beans and passes the received class to the appropriate document service using a registry program, for example using the event listener architecture described above. The document service 2104 which receives the document in the form of JAVA beans from the router 2103 acts as the interface to the enterprise solution software. This includes a registry service 2110 by which listeners to XML events are coupled with the incoming data streams, and a service manager 2111 to manage the routing of the incoming documents to the appropriate services. The document service manager 2111 provides for administration of the registry service and for maintaining document consistency and the like.

The document service communicates with the back end system using any proprietary API, or using such more common forms as the CORBA/COM interface or other architectures.

Fig. 22 provides a heuristic diagram of the market maker and market participant structures according to the present invention. This model has been largely adopted by eCo as a standard. Thus, the electronic commerce market according to the present invention can be logically organized as set forth in Fig. 22. At the top of the organization, a market maker node 2200 is established. The market maker node includes resources that establish a marketplace 2201. Such resources include a market registry service and the like. Businesses 2202 register in the marketplace 2201 by publishing a business interface definition. The business interface definition defines the services 2203 for commercial transactions in which the businesses will participate. The transactions 2204 and services 2203 use documents 2205 to define the inputs and outputs, and outline the commercial relationship between participants in the transaction. The documents have content 2206 which carries the particulars of each transaction. The manner in which the content is processed by the participants in the market, and by the market maker is completely independent of the document based electronic commerce network which is established according to the present invention. Overall, a robust, scalable, intuitive structure is presented for enabling electronic commerce on communication networks is provided.

Thus, the present invention in an exemplary system provides a platform based on the XML processor and uses XML documents as the interface between loosely coupled

business systems. The documents are transferred between businesses and processed by participant nodes before entering the company business system. Thus the platform enables electronic commerce applications between businesses where each business system operates using different internal commerce platforms, processes and semantics, by specifying a common set of business documents and forms.

According to the present invention, virtual enterprises are created by interconnecting business systems and service, are primarily defined in terms of the documents (XML-encoded) that businesses accept and generate:

- "if you send me a request for a catalog, I will send you a catalog:
- "if you send me a purchase order and I can accept it, I will send you an invoice".

The foregoing description of an exemplary embodiment of the invention has been presented for purposes of illustration and description. It is not intended to be exhaustive or to limit the invention to the precise forms disclosed. Obviously, many modifications and variations will be apparent to practitioners skilled in this art. It is intended that the scope of the invention be defined by the following claims and their equivalents.

What is claimed is:

# CLAIMS

1. A registry supporting transactions among a plurality of participants in a network including one or more market nodes and a plurality of participant nodes, comprising, a machine readable registry accessible to at least one participant node with entries for

traders utilizing participant nodes;

service processes executing on the market nodes;

terms and conditions applicable to at least a portion of the transactions among the participant nodes; and

a machine readable specification of an interface to transaction processes stored in memory accessible by at least one participant node in the network, including interpretation information providing a definition of an input document, and a definition of an output document.

2. The registry of claim 1, wherein the entries comprise storage units and logical structures for the sets of storage units.

3. The registry of claim 1, wherein the respective definitions of input and output documents comprise storage units and logical structures for the sets of storage units.

4. The registry of claim 3, wherein the interpretation information includes at least one data structure mapping predefined sets of storage units for a particular logical structure in the definitions of the input and output documents, to respective elements in a list.

5. The registry of claim 3, including a repository in memory accessible by at least one participant node storing a library of logical structures, and interpretation information for logic structures.

6. The registry of claim 3, wherein the machine readable specification includes a document compliant with a definition of an interface document including logical structures for storing an identifier of a particular transaction, and at least one of

definitions and references to definitions of input and output documents for the particular transaction.

7. The registry of claim 3, wherein the machine readable specification includes a document compliant with a definition of an interface document including logical structures for storing an identifier of the interface, and for storing at least one of specifications and references to specifications of a set of one or more transactions supported by the interface.

8. The registry of claim 7, wherein the machine readable specification includes a reference to a specification of a particular transaction, and the specification of the particular transaction includes a document including logical structures for storing at least one of definitions and references to definitions of input and output documents for the particular transaction.

9. The registry of claim 3, wherein the storage units comprise parsed data.

10. The registry of claim 9, wherein the parsed data in at least one of the input and output documents comprises:

character data encoding text characters in the one of the input and output documents, and

markup data identifying sets of storage units according to the logical structure of the one of the input and output documents.

11. The registry of claim 10, wherein at least one of the sets of storage units encodes a plurality of text characters providing a natural language word.

12. The registry of claim 9, wherein the interpretation information for at least one of the sets of storage units identified by a particular logical structure of at least one of the input and output documents, encodes respective definitions for sets of parsed characters.

13. The registry of claim 9, wherein the storage units comprise unparsed data.

14. The registry of claim 3, including a repository stored in memory accessible by at least one node in the network of document types for use in a plurality of transactions, and wherein the definition of one of the input and output documents includes a reference to a document type in the repository.

15. The registry of claim 14, wherein the repository of document types includes a document type for identifying participant processes in the network.

16. The registry of claim 3, wherein the definitions of the input and output documents comprise document type definitions compliant with a standard Extensible Markup Language XML.

17. The registry of claim 3, wherein the machine readable data structure including interpretation information comprises a document organized according to a document type definition compliant with a standard Extensible Markup Language XML.

18. The registry of claim 1, wherein the entries for traders are associated with roles including buyer, supplier and service provider.

19. The registry of claim 1, wherein the entries for traders are associated with identifiers of organizations or natural persons.

20. The registry of claim 1, wherein the entries for traders are associated with identifiers of organizations or natural persons and with applications executing on the participant nodes.

21. The registry of claim 1, wherein the entries for traders are associated with identifiers of natural persons having roles including operator, technical contact, and administrative contact.

22. The registry of claim 1, wherein the entries for service processes are associated with roles including system services, business services, portal services and community services.

23. The registry of claim 1, wherein the entries for service processes are associated with roles including system services, business services, portal services and community services, the respective services being identified by universal resource names.

24. The registry of claim 1, wherein the entries for service processes are associated with roles including system services, business services, portal services and community services, the respective services being identified by universal resource locators.

25. The registry of claim 1, wherein the registry entries comprise data received from the participant nodes and cached in memory.

26. The registry of claim 1, wherein the registry entries comprise meta data references to one or more locations where data is available.

27. A registry supporting transactions among a plurality of participants in a network including a plurality of market nodes and a plurality of participant nodes, comprising a machine readable multi-market registry accessible to at least one participant node with entries for

the market nodes;

traders utilizing participant nodes;

service processes executing on the market nodes;

terms and conditions applicable to at least a portion of the transactions among the participant nodes; and

a machine readable specification of an interface to transaction processes stored in memory accessible by at least one participant node in the network, including interpretation information providing a definition of an input document, and a definition of an output document.

28. The registry of claim 27, wherein the respective definitions of input and output documents comprise storage units and logical structures for the sets of storage units.

29. The registry of claim 28, wherein the interpretation information includes at least one data structure mapping predefined sets of storage units for a particular logical structure in the definitions of the input and output documents, to respective elements in a list.

30. The registry of claim 28, including a repository in memory accessible by at least one node in the network storing a library of logical structures, and interpretation information for logic structures.

31. The registry of claim 28, wherein the machine readable specification includes a document compliant with a definition of an interface document including logical structures for storing an identifier of a particular transaction, and at least one of definitions and references to definitions of input and output documents for the particular transaction.

32. The registry of claim 28, wherein the machine readable specification includes a document compliant with a definition of an interface document including logical structures for storing an identifier of the interface, and for storing at least one of specifications and references to specifications of a set of one or more transactions supported by the interface.

33. The registry of claim 32, wherein the machine readable specification includes a reference to a specification of a particular transaction, and the specification of the particular transaction includes a document including logical structures for storing at least one of definitions and references to definitions of input and output documents for the particular transaction.

34. The registry of claim 28, wherein the storage units comprise parsed data.

35. The registry of claim 34, wherein the parsed data in at least one of the input and output documents comprises:

character data encoding text characters in the one of the input and output documents, and

markup data identifying sets of storage units according to the logical structure of the one of the input and output documents.

36. The registry of claim 32, wherein the storage units comprise unparsed data.

37. The registry of claim 28, including a repository stored in memory accessible by at least one node in the network of document types for use in a plurality of transactions, and wherein the definition of one of the input and output documents includes a reference to a document type in the repository.

38. The registry of claim 37, wherein the repository of document types includes a document type for identifying participant processes in the network.

39. The registry of claim 28, wherein the definitions of the input and output documents comprise document type definitions compliant with a standard Extensible Markup Language XML.

40. The registry of claim 28, wherein the machine readable data structure including interpretation information comprises a document organized according to a document type definition compliant with a standard Extensible Markup Language XML.

41. The registry of claim 27, wherein the entries for traders are associated with roles including buyer, supplier and service provider.

42. The registry of claim 27, wherein the entries for traders are associated with identifiers of organizations or natural persons.

43. The registry of claim 27, wherein the entries for traders are associated with identifiers of natural persons having roles including operator, technical contact, and administrative contact.

44. The registry of claim 27, wherein the entries for service processes are associated with roles including system services, business services, portal services and community services.

45. The registry of claim 27, wherein the entries for service processes are associated with roles including system services, business services, portal services and community services, the respective services being identified by universal resource names.

46. The registry of claim 27, wherein the entries for service processes are associated with roles including system services, business services, portal services and community services, the respective services being identified by universal resource locators.

47. The registry of claim 27, wherein the multi-market registry entries comprise data received from the participant nodes and cached in memory.

48. The registry of claim 27, wherein the multi-market registry entries comprise meta data references to one or more locations where data is available.

49. The registry of claim 27, wherein the multi-market registry entries comprise data replicated from the market nodes.

50. A method for executing transactions among participant nodes in a network including a root node, a plurality of market nodes and a plurality of participant nodes which execute processes involved in the transactions, comprising:

submitting a machine-readable specification of an interface for a transaction to a market node, the specification including a definition of an input document, and a definition of an output document;

receiving data comprising a document through a communication network;

parsing the document according to the specification to identify an input document;

providing at least a portion of the input document in a machine-readable format to a transaction process which produces an output;

forming, based on the specification, an output document comprising the output according to the definition of an output document; and

transmitting the output document on the communication network.

51. The method of claim 50, wherein the definitions of the input and output documents comprising respective descriptions of sets of storage units and logical structures for the sets of storage units.

52. The method of claim 50, wherein the document is received by a participant node directly from an additional participant node.

53. The method of claim 52, further including reporting to a market node communication of the document between the participant node and the additional participant node.

54. The method of claim 53, wherein the document creates a binding obligation, further comprising reporting to the market node the binding obligation.

55. The method of claim 53, wherein the document creates a binding obligation, further comprising applying stored terms and conditions to the binding obligation.

56. A method for executing transactions among a plurality of participants in a network including a root node, a plurality of market nodes and a plurality of participant nodes, comprising:

accessing at least one of a root node or of a plurality of market nodes to obtain registry information regarding a trader, including a location;

sending the trader, through the network to the location, data comprising a document, said document conforming to a machine-readable specification of an interface

for a transaction, the specification including a definition of an input document, and a definition of an output document, the definitions of the input and output documents comprising respective descriptions of sets of storage units and logical structures for the sets of storage units;

receiving data comprising an additional document through a communication network;

parsing the additional document according to the specification;

providing at least a portion of the parsed data in a machine-readable format to a transaction process which produces an output;

forming, based on the specification, an output document comprising the output according to the definition of an output document; and

transmitting the output document on the communication network.

57. The method of claim 56, further including:

accessing a specification of a complementary interface provided for another node in the network for the transaction, the accessed specification including a definition of an input document for the complementary interface, and a definition of an output document for the complementary interface; and

establishing the stored specification of the interface by including at least part of the definition of the output document of the complementary interface in the definition of the input document of the interface in the stored specification.

58. The method of claim 57, including:

finding the complementary interface in the network.

59. The method of claim 57, wherein the establishing the stored specification includes accessing elements of the machine-readable specification from a repository, the repository storing a library of logical structures, schematic maps for logic structures, and definitions of documents comprising logic structures used to build interface descriptions.

60. The method of claim 57, including:

establishing the stored specification of the interface by including at least part of the definition of the input document of the complementary interface in the definition of the output document of the interface in the stored specification.

61. The method of claim 56, including providing access to the specification through the communication network to other nodes in the network.

62. The method of claim 56, including sending the specification of the interface to another node in the network, at which access to the specification is provided for other nodes in the network.

63. The method of claim 56, wherein the machine-readable specification includes a document compliant with a definition of an interface document including logical structures for storing an identifier of a particular transaction, and at least one of definitions and references to definitions of input and output documents for the particular transaction.

64. The method of claim 56, wherein the machine-readable specification includes a document compliant with a definition of an interface document including logical structures for storing an identifier of the interface, and for storing at least one of specifications and references to specifications of a set of one or more transactions supported by the interface.

65. The method of claim 64, wherein the machine-readable specification includes a reference to a specification of a particular transaction, and the specification of the particular transaction includes a document including logical structures for storing at least one of definitions and references to definitions of input and output documents for the particular transaction.

66. The method of claim 57, wherein the storage units comprise parsed data.

67. The method of claim 66, wherein the parsed data in at least one of the input and output documents comprises:

character data encoding text characters in the one of the input and output documents, and

markup data identifying sets of storage units according to the logical structure of the one of the input and output documents.

68. The method of claim 67, wherein at least one of the sets of storage units encodes a plurality of text characters providing a natural language word.

69. The method of claim 66, wherein the specification includes interpretation information for at least one of the sets of storage units identified by the logical structure of at least one of the input and output documents, encoding respective definitions for sets of parsed characters.

70. The method of claim 66, wherein the storage units comprise unparsed data.

71. The method of claim 56, wherein the transaction process has a variant transaction processing architecture, and including translating at least of portion of the input document into a format readable according to the variant transaction processing architecture of the transaction process.

72. The method of claim 71, wherein the translating includes producing programming objects including variables and methods according to the variant transaction processing architecture of the transaction process.

73. The method of claim 71, wherein the variant transaction processing architecture of the transaction process includes a process compliant with an interface description language.

74. The method of claim 56, wherein the definitions of the input and output documents comprise document type definitions compliant with a standard Extensible Markup Language XML.

# Global Trading Web Architecture

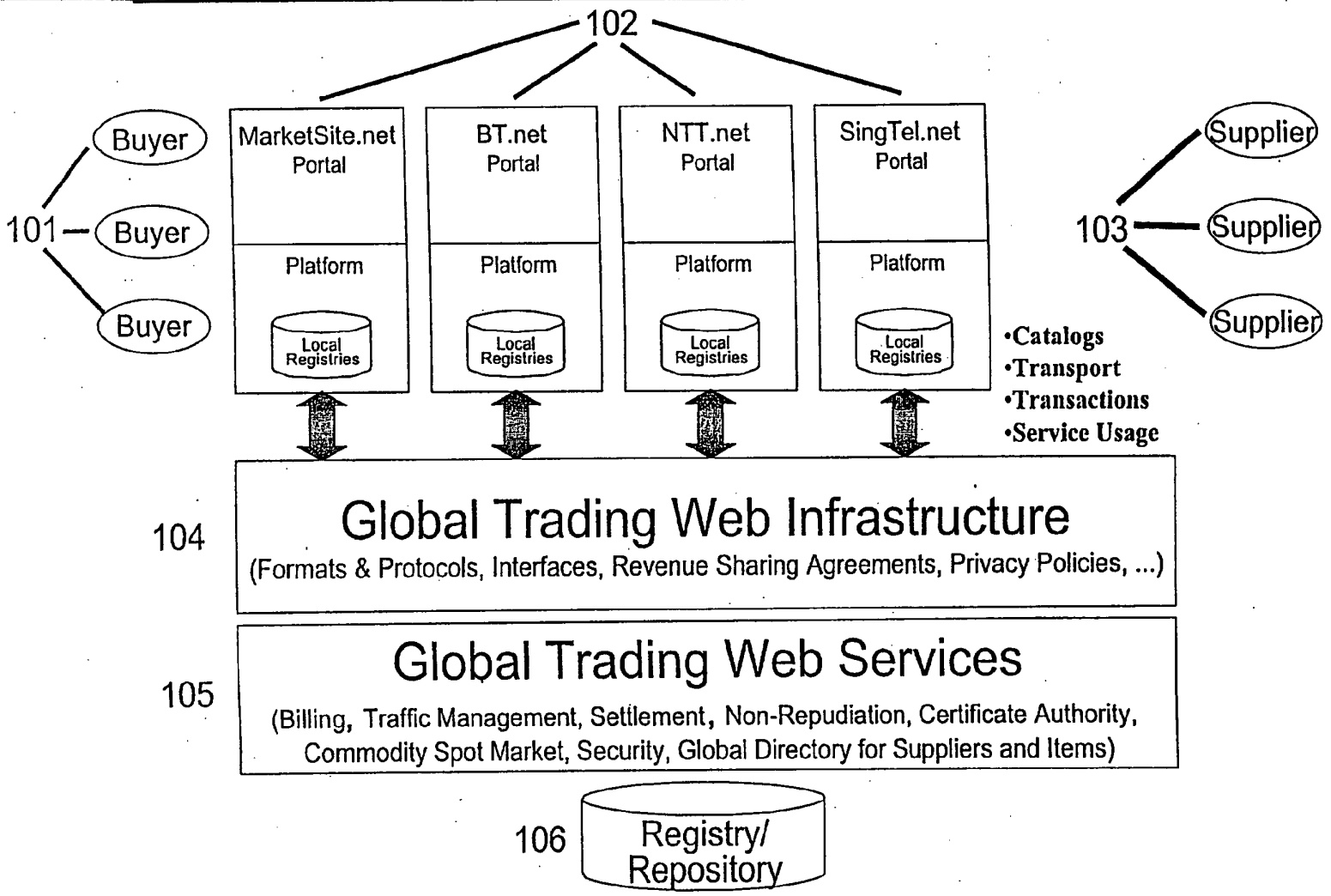



Figure 1

COMMERCE ONE

# GTW Topology

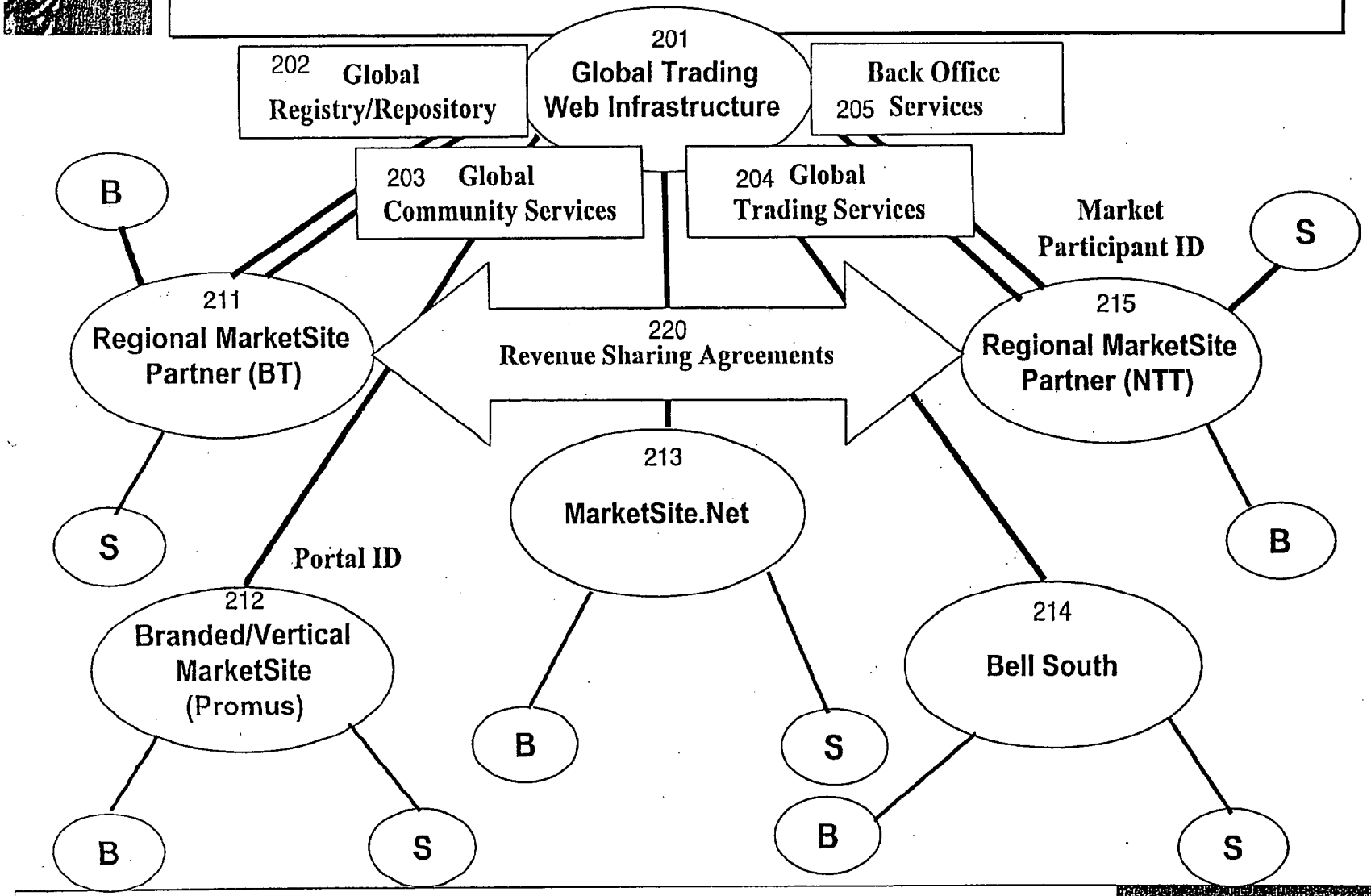



Figure 2

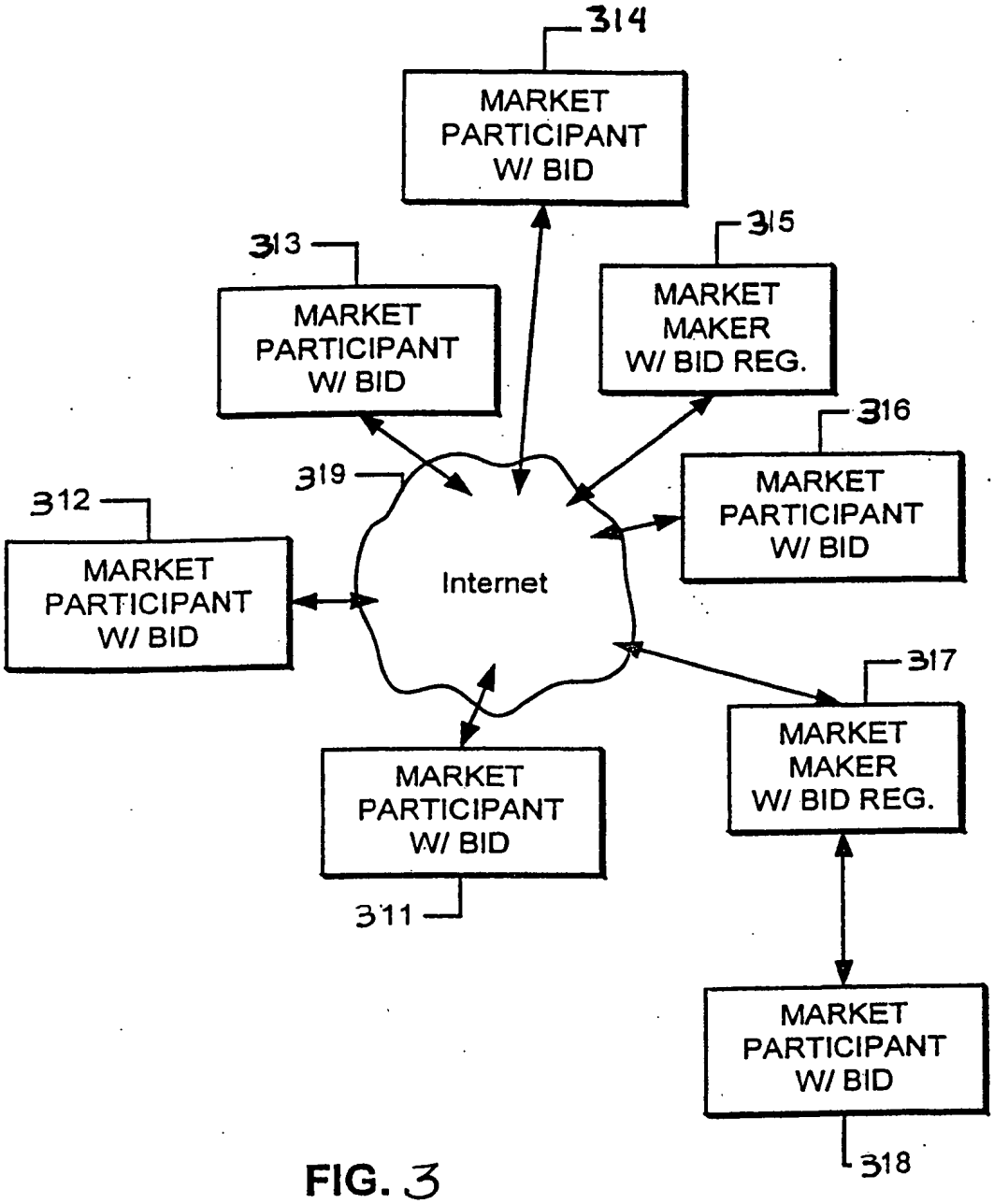

314
MARKET PARTICIPANT W/ BID
313
MARKET PARTICIPANT W/ BID
315
MARKET MAKER W/ BID REG.
316
MARKET PARTICIPANT W/ BID
319
Internet
312
MARKET PARTICIPANT W/ BID
317
MARKET MAKER W/ BID REG.
MARKET PARTICIPANT W/ BID
311
MARKET PARTICIPANT W/ BID
318

FIG. 3

# GTW Implementation Option

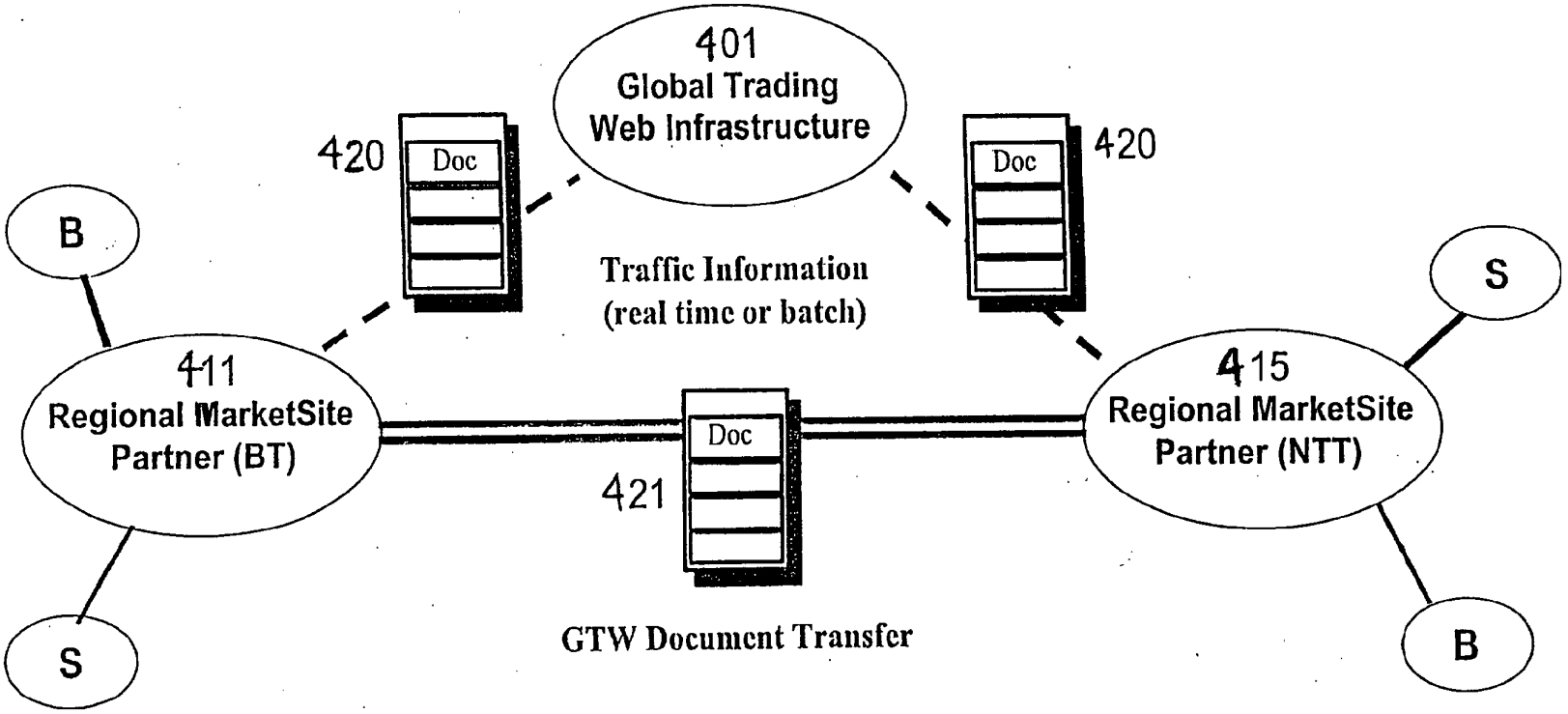



Figure 4

COMMERCE ONE

# Registry/Repository High Level Architecture

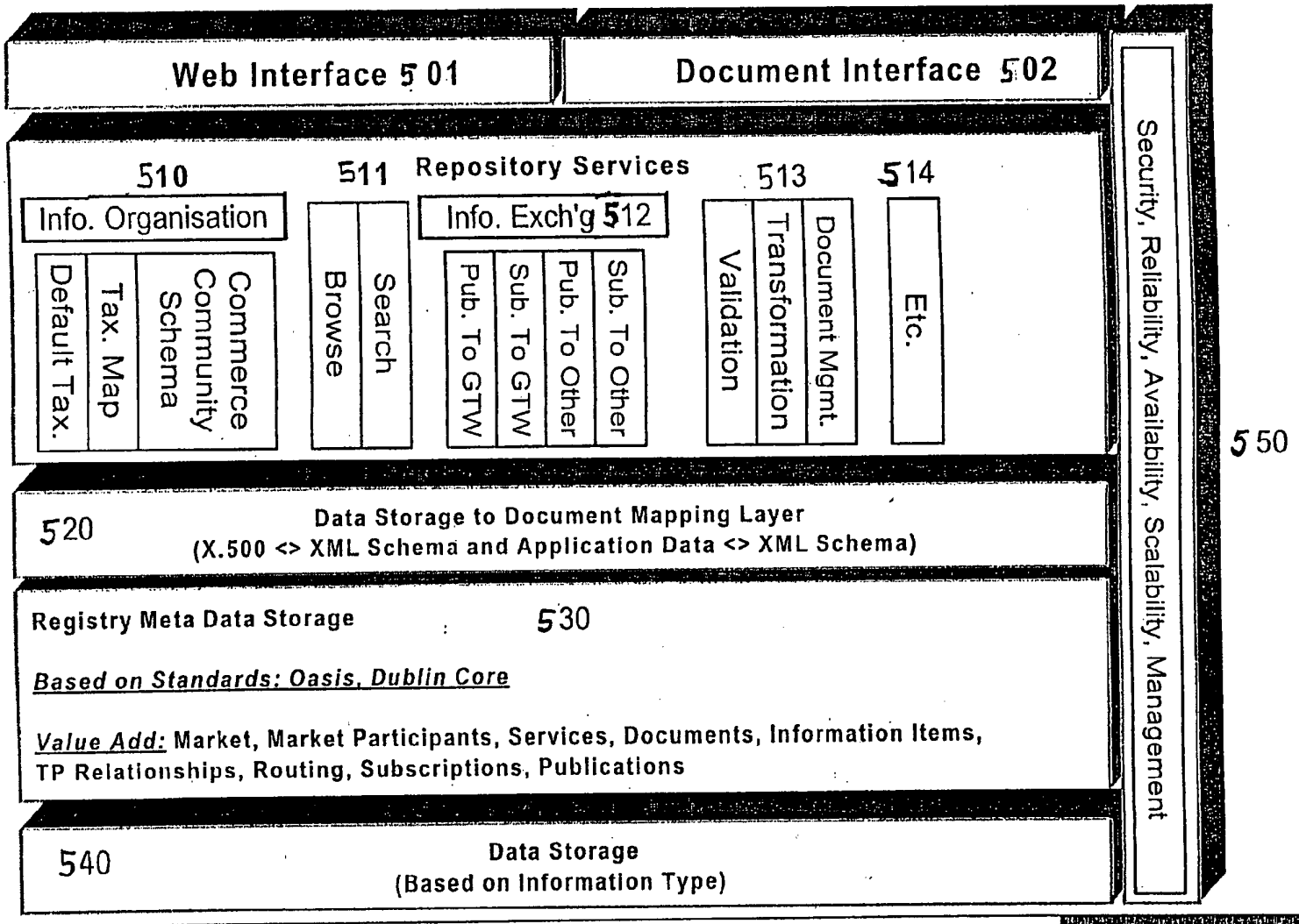



Figure 5

COMMERCE ONE

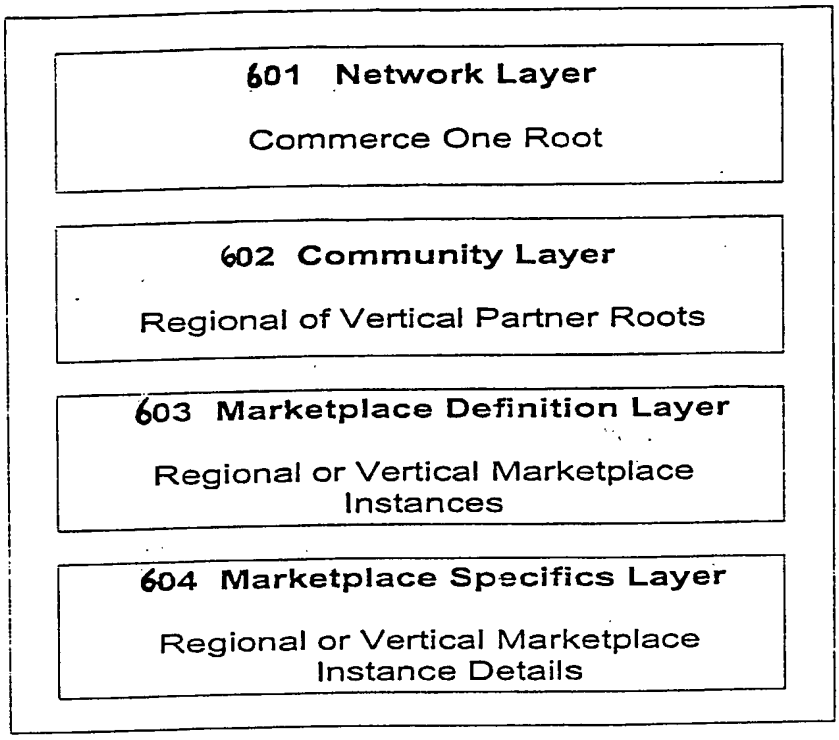



**Figure 6**- Commerce Community Schema Entity and Relationship Diagram

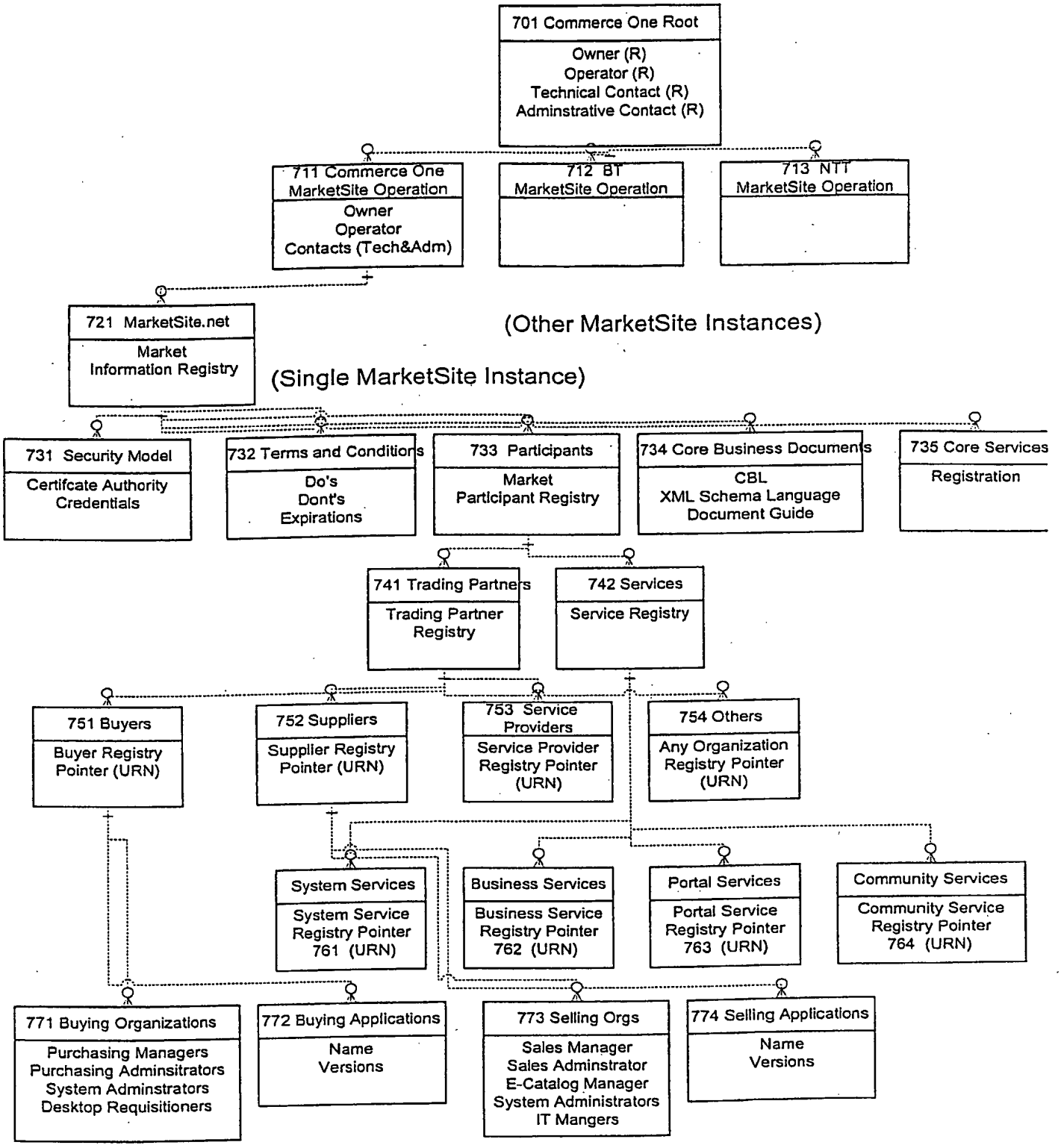
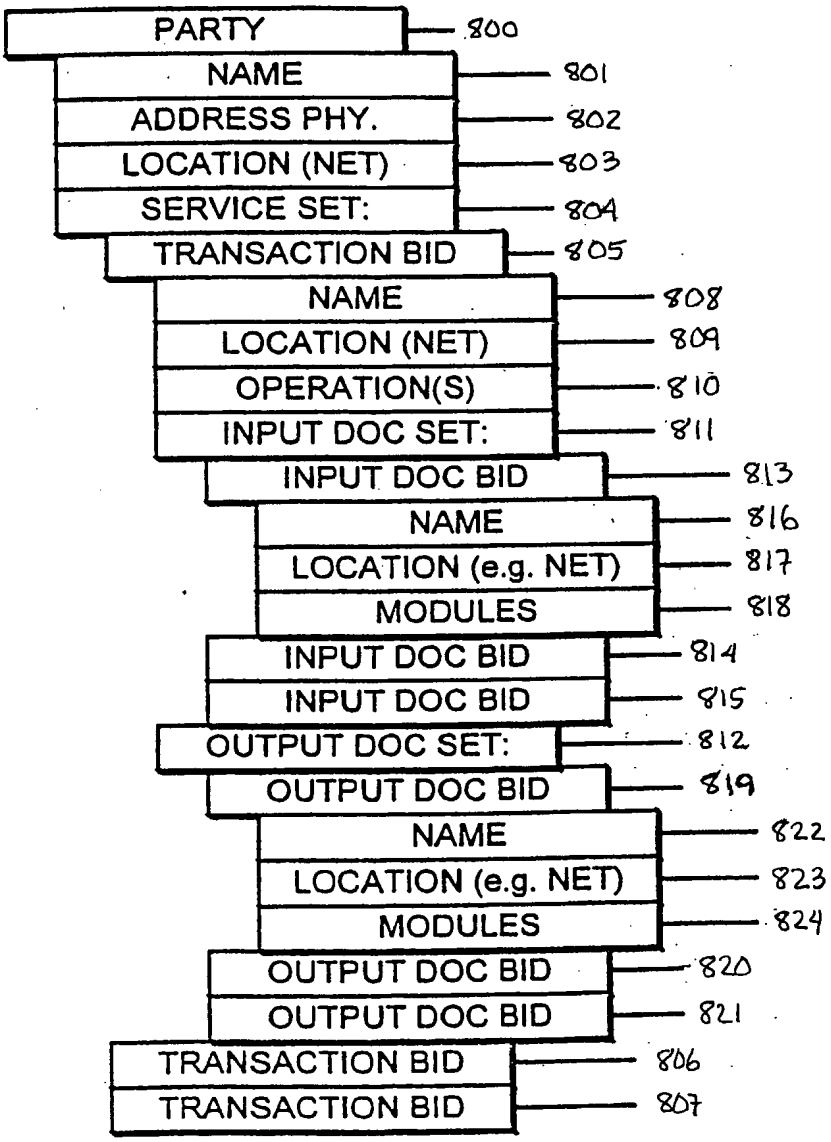

701 Commerce One Root
Owner (R)
Operator (R)
Technical Contact (R)
Adminstrative Contact (R)
711 Commerce One MarketSite Operation
Owner
Operator
Contacts (Tech&Adm)
712 BT MarketSite Operation
713 NTT MarketSite Operation
(Other MarketSite Instances)
721 MarketSite.net
Market
Information Registry
(Single MarketSite Instance)
731 Security Model
Certifcate Authority
Credentials
732 Terms and Conditions
Do's
Dont's
Expirations
733 Participants
Market
Participant Registry
734 Core Business Documents
CBL
XML Schema Language
Document Guide
735 Core Services
Registration
741 Trading Partners
Trading Partner
Registry
742 Services
Service Registry
751 Buyers
Buyer Registry
Pointer (URN)
752 Suppliers
Supplier Registry
Pointer (URN)
753 Service Providers
Service Provider
Registry Pointer
(URN)
754 Others
Any Organization
Registry Pointer
(URN)
System Services
System Service
Registry Pointer
761 (URN)
Business Services
Business Service
Registry Pointer
762 (URN)
Portal Services
Portal Service
Registry Pointer
763 (URN)
Community Services
Community Service
Registry Pointer
764 (URN)
771 Buying Organizations
Purchasing Managers
Purchasing Adminsitrators
System Adminstrators
Desktop Requisitioners
772 Buying Applications
Name
Versions
773 Selling Orgs
Sales Manager
Sales Adminstrator
E-Catalog Manager
System Administrators
IT Mangers
774 Selling Applications
Name
Versions

Figure 7

FIG. 8


PARTY
800
NAME
801
ADDRESS PHY.
802
LOCATION (NET)
803
SERVICE SET:
804
TRANSACTION BID
805
NAME
808
LOCATION (NET)
809
OPERATION(S)
810
INPUT DOC SET:
811
INPUT DOC BID
813
NAME
816
LOCATION (e.g. NET)
817
MODULES
818
INPUT DOC BID
814
INPUT DOC BID
815
OUTPUT DOC SET:
812
OUTPUT DOC BID
819
NAME
822
LOCATION (e.g. NET)
823
MODULES
824
OUTPUT DOC BID
820
OUTPUT DOC BID
821
TRANSACTION BID
806
TRANSACTION BID
807

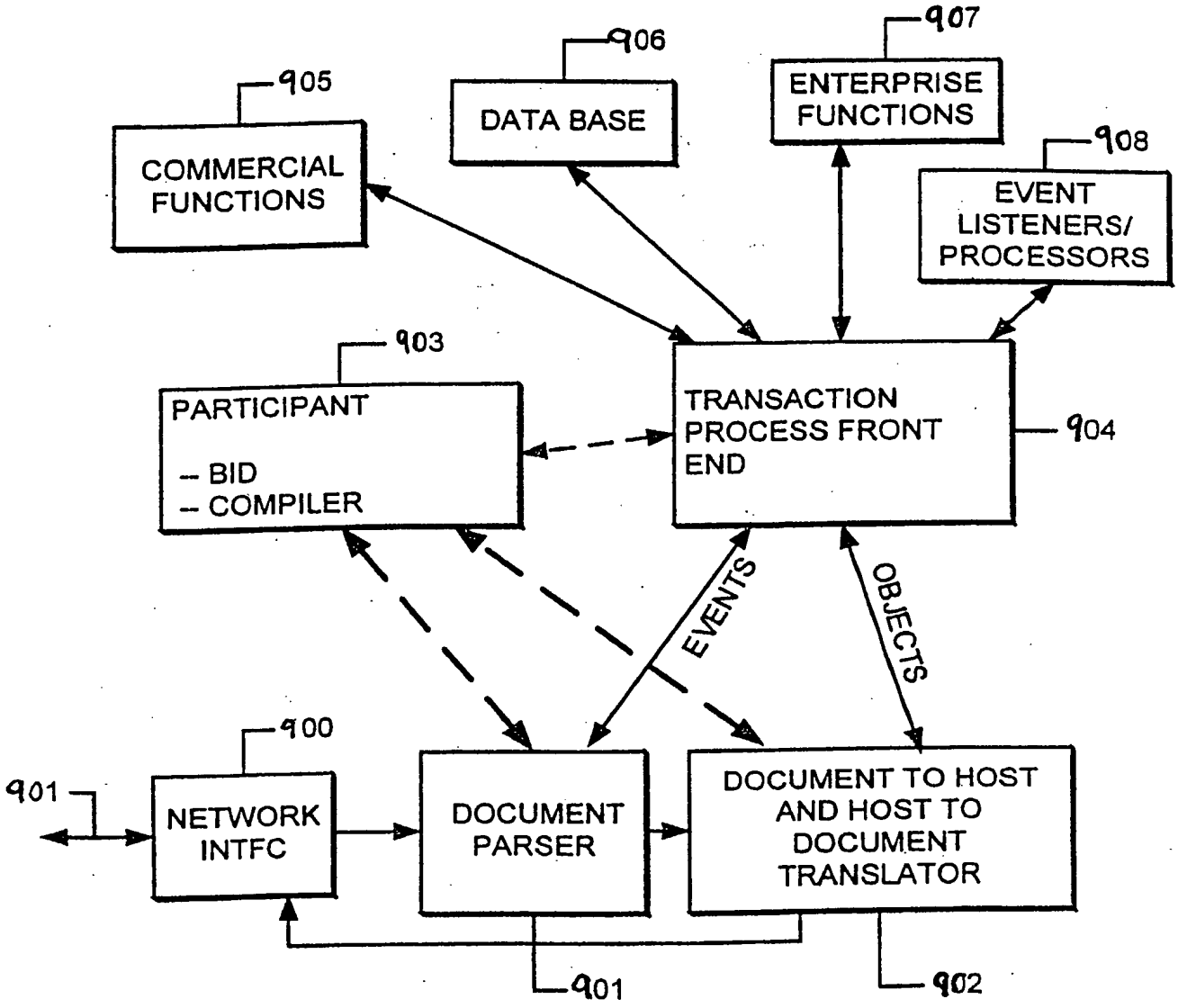

905
COMMERCIAL FUNCTIONS
906
DATA BASE
907
ENTERPRISE FUNCTIONS
908
EVENT LISTENERS/ PROCESSORS
903
PARTICIPANT
– BID
– COMPILER
TRANSACTION PROCESS FRONT END
904
EVENTS
OBJECTS
900
901
NETWORK INTFC
DOCUMENT PARSER
DOCUMENT TO HOST AND HOST TO DOCUMENT TRANSLATOR
901
902

FIG. 9

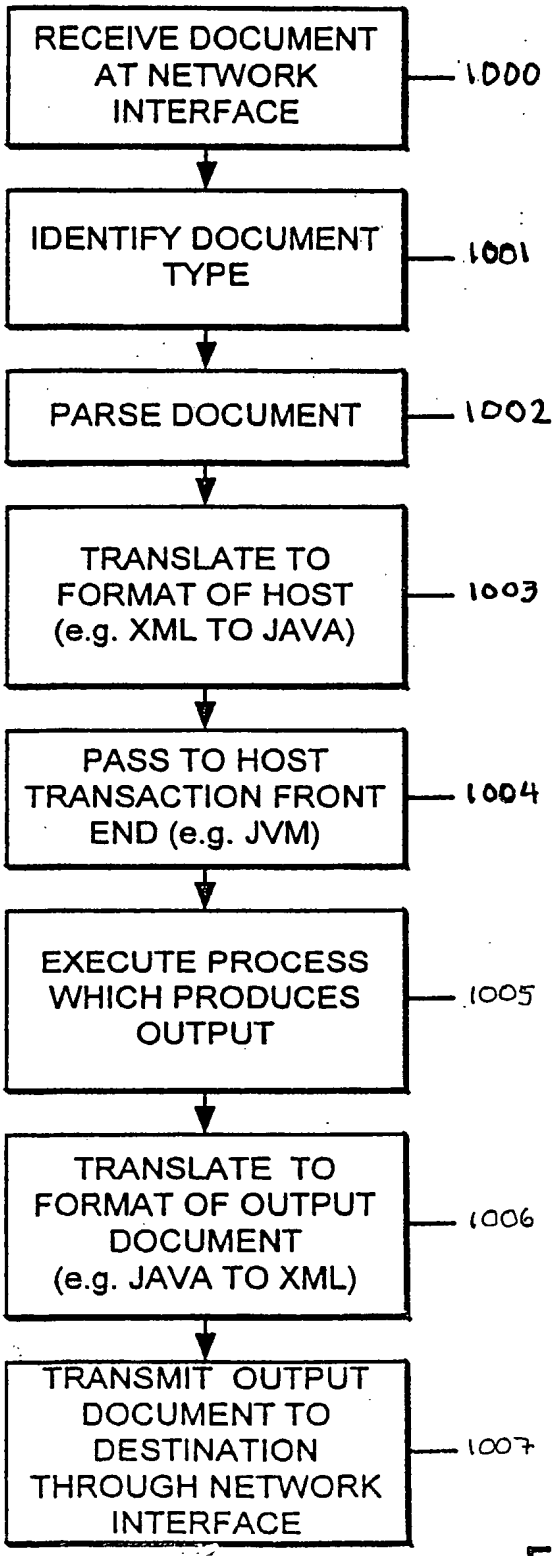

RECEIVE DOCUMENT AT NETWORK INTERFACE
1000
IDENTIFY DOCUMENT TYPE
1001
PARSE DOCUMENT
1002
TRANSLATE TO FORMAT OF HOST (e.g. XML TO JAVA)
1003
PASS TO HOST TRANSACTION FRONT END (e.g. JVM)
1004
EXECUTE PROCESS WHICH PRODUCES OUTPUT
1005
TRANSLATE TO FORMAT OF OUTPUT DOCUMENT (e.g. JAVA TO XML)
1006
TRANSMIT OUTPUT DOCUMENT TO DESTINATION THROUGH NETWORK INTERFACE
1007

FIG. 10

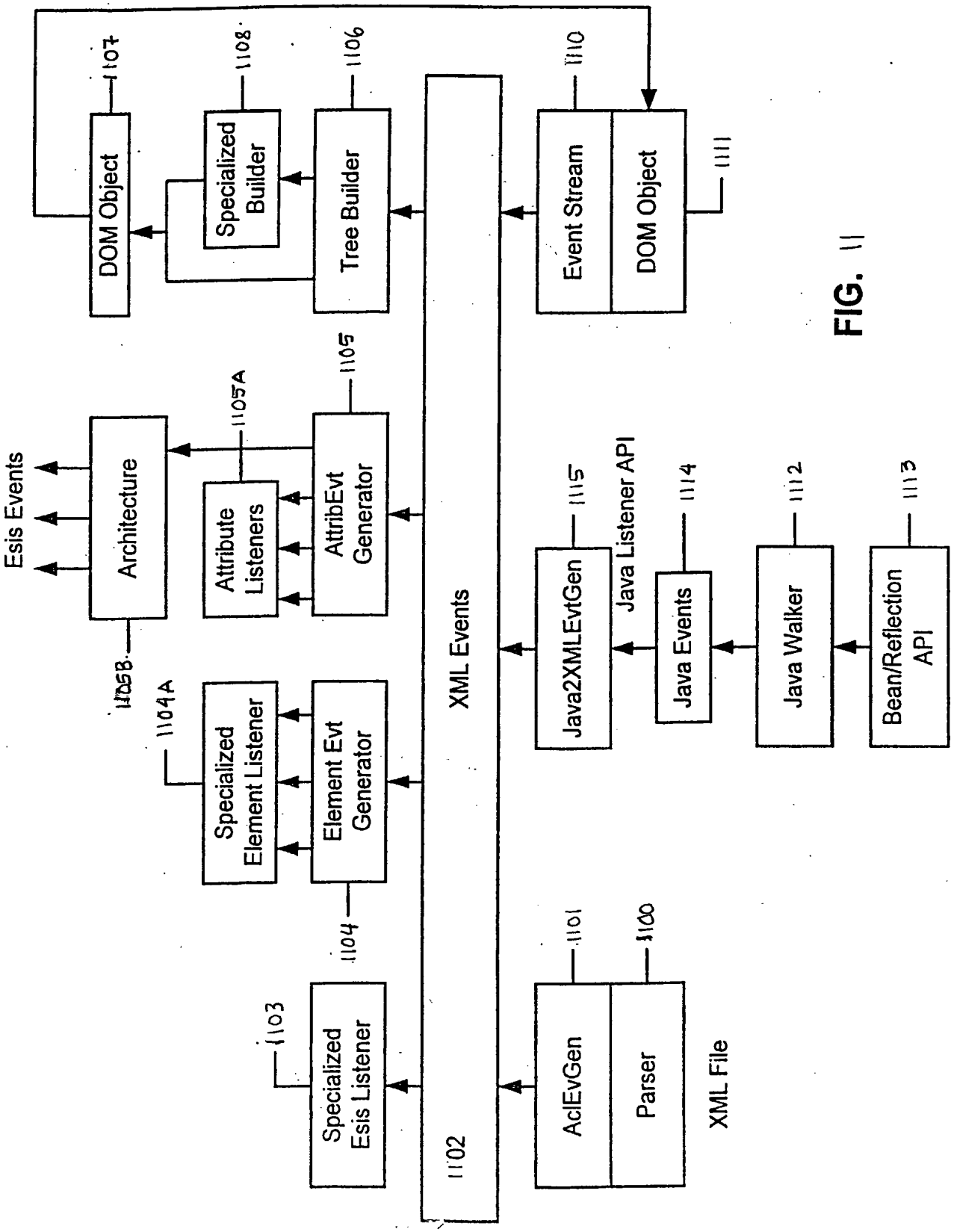

DOM Object
1107
Specialized Builder
1108
Tree Builder
1106
Event Stream
1110
DOM Object
1111
Esis Events
Architecture
1105A
Attribute Listeners
1105
AttribEvt Generator
1105B
1104A
Specialized Element Listener
Element Evt Generator
1104
XML Events
Java2XMLEvtGen
1115
Java Listener API
Java Events
1114
Java Walker
1112
Bean/Reflection API
1113
1103
Specialized Esis Listener
1102
AclEvGen
1101
Parser
1100
XML File

FIG. 11

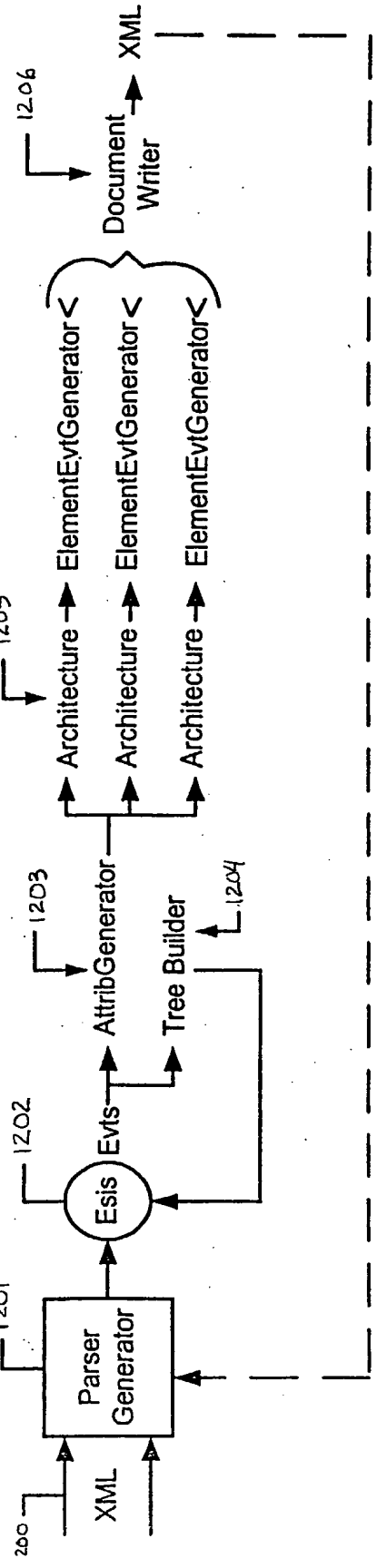

200
XML
1201
Parser Generator
1202
Esis
Evts
1203
AttribGenerator
Tree Builder
1204
1205
Architecture
ElementEvtGenerator
Architecture
ElementEvtGenerator
Architecture
ElementEvtGenerator
1206
Document Writer
XML

FIG. 12

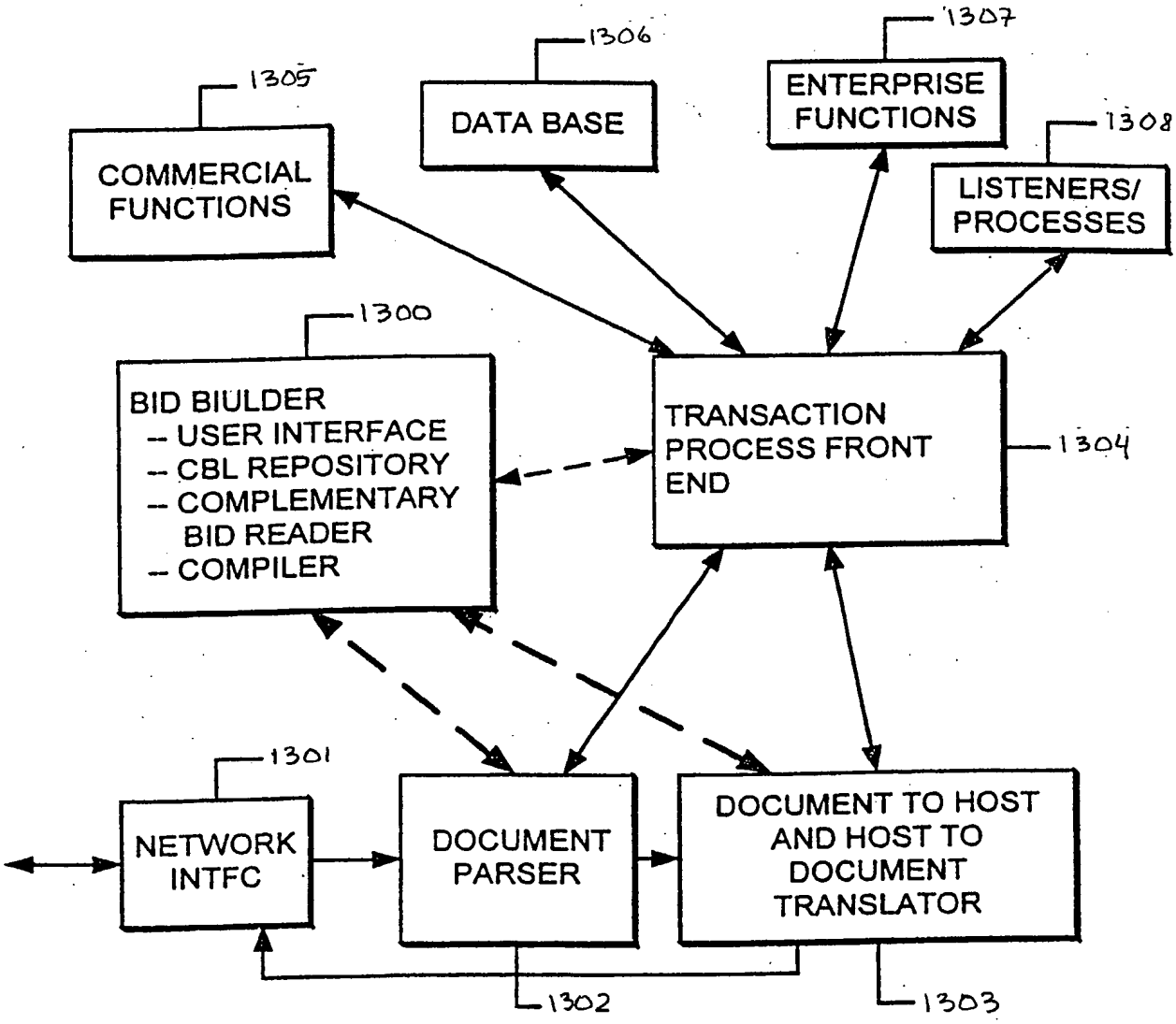

1305
COMMERCIAL FUNCTIONS
1306
DATA BASE
1307
ENTERPRISE FUNCTIONS
1308
LISTENERS/ PROCESSES
1300
BID BIULDER
-- USER INTERFACE
-- CBL REPOSITORY
-- COMPLEMENTARY BID READER
-- COMPILER
TRANSACTION PROCESS FRONT END
1304
1301
NETWORK INTFC
DOCUMENT PARSER
DOCUMENT TO HOST AND HOST TO DOCUMENT TRANSLATOR
1302
1303

FIG. 13

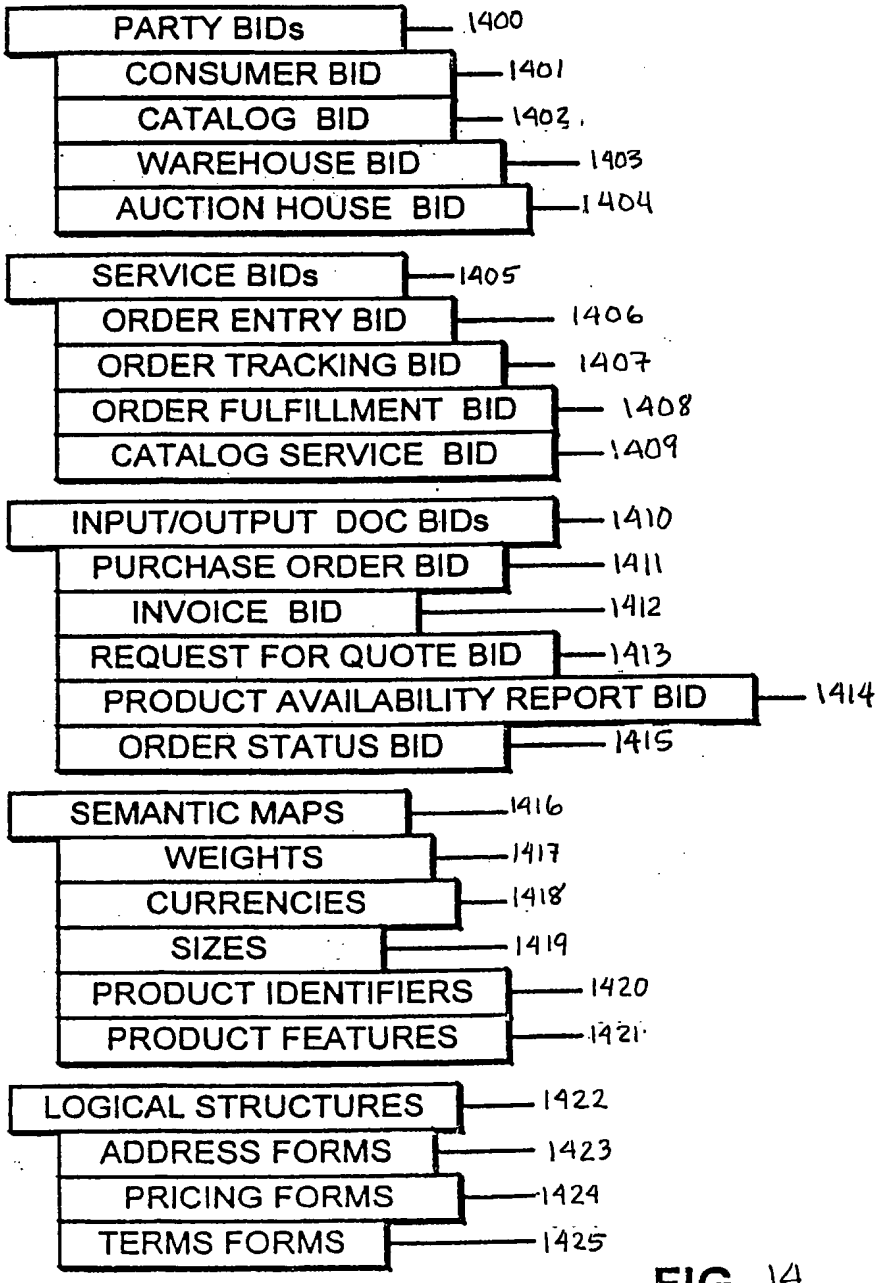

PARTY BIDs
1400
CONSUMER BID
1401
CATALOG BID
1402
WAREHOUSE BID
1403
AUCTION HOUSE BID
1404
SERVICE BIDs
1405
ORDER ENTRY BID
1406
ORDER TRACKING BID
1407
ORDER FULFILLMENT BID
1408
CATALOG SERVICE BID
1409
INPUT/OUTPUT DOC BIDs
1410
PURCHASE ORDER BID
1411
INVOICE BID
1412
REQUEST FOR QUOTE BID
1413
PRODUCT AVAILABILITY REPORT BID
1414
ORDER STATUS BID
1415
SEMANTIC MAPS
1416
WEIGHTS
1417
CURRENCIES
1418
SIZES
1419
PRODUCT IDENTIFIERS
1420
PRODUCT FEATURES
1421
LOGICAL STRUCTURES
1422
ADDRESS FORMS
1423
PRICING FORMS
1424
TERMS FORMS
1425

FIG. 14

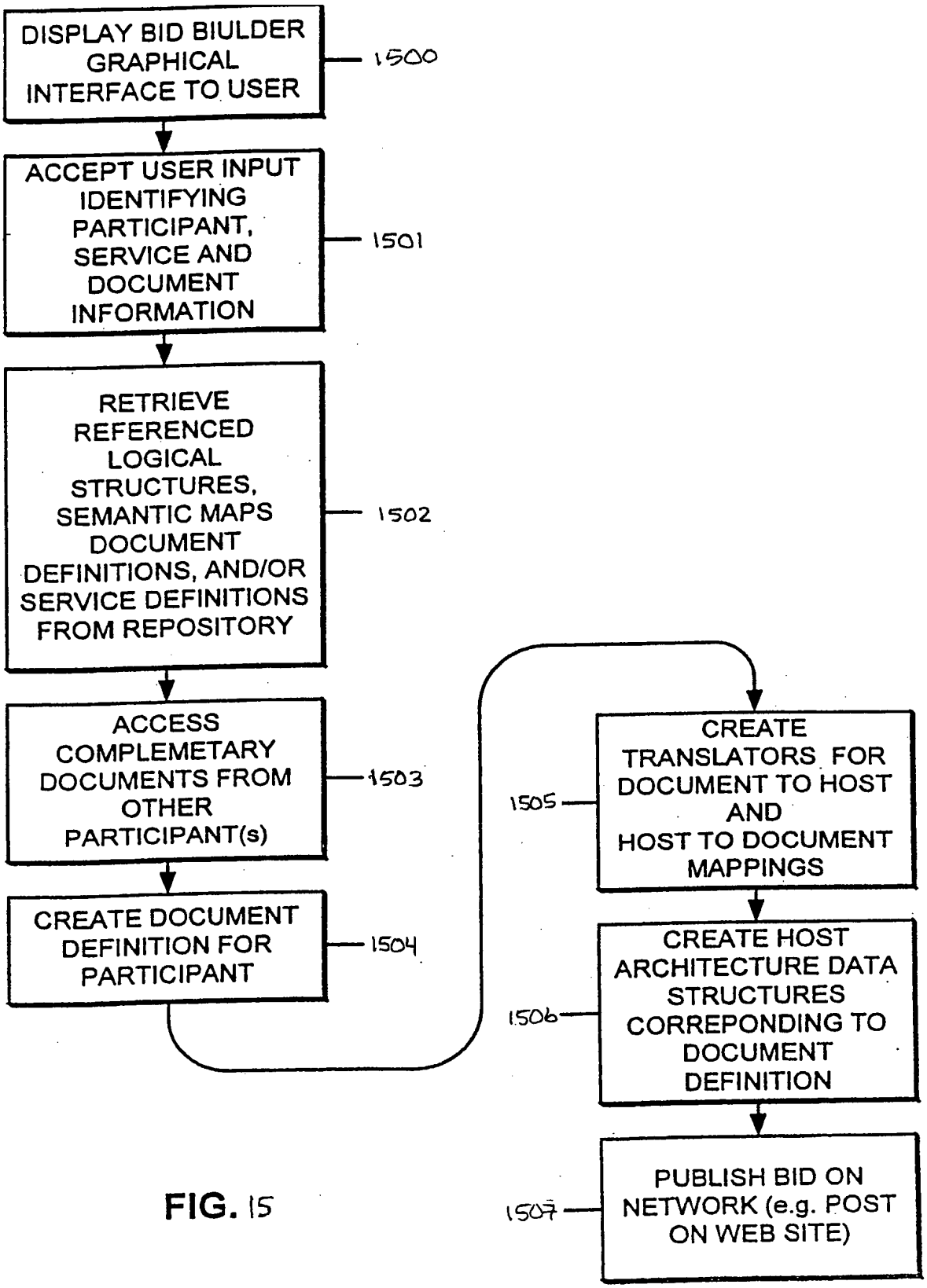

DISPLAY BID BIULDER GRAPHICAL INTERFACE TO USER
1500
ACCEPT USER INPUT IDENTIFYING PARTICIPANT, SERVICE AND DOCUMENT INFORMATION
1501
RETRIEVE REFERENCED LOGICAL STRUCTURES, SEMANTIC MAPS DOCUMENT DEFINITIONS, AND/OR SERVICE DEFINITIONS FROM REPOSITORY
1502
ACCESS COMPLEMETARY DOCUMENTS FROM OTHER PARTICIPANT(s)
1503
CREATE DOCUMENT DEFINITION FOR PARTICIPANT
1504
1505
CREATE TRANSLATORS FOR DOCUMENT TO HOST AND HOST TO DOCUMENT MAPPINGS
1506
CREATE HOST ARCHITECTURE DATA STRUCTURES CORREPONDING TO DOCUMENT DEFINITION
1507
PUBLISH BID ON NETWORK (e.g. POST ON WEB SITE)

FIG. 15

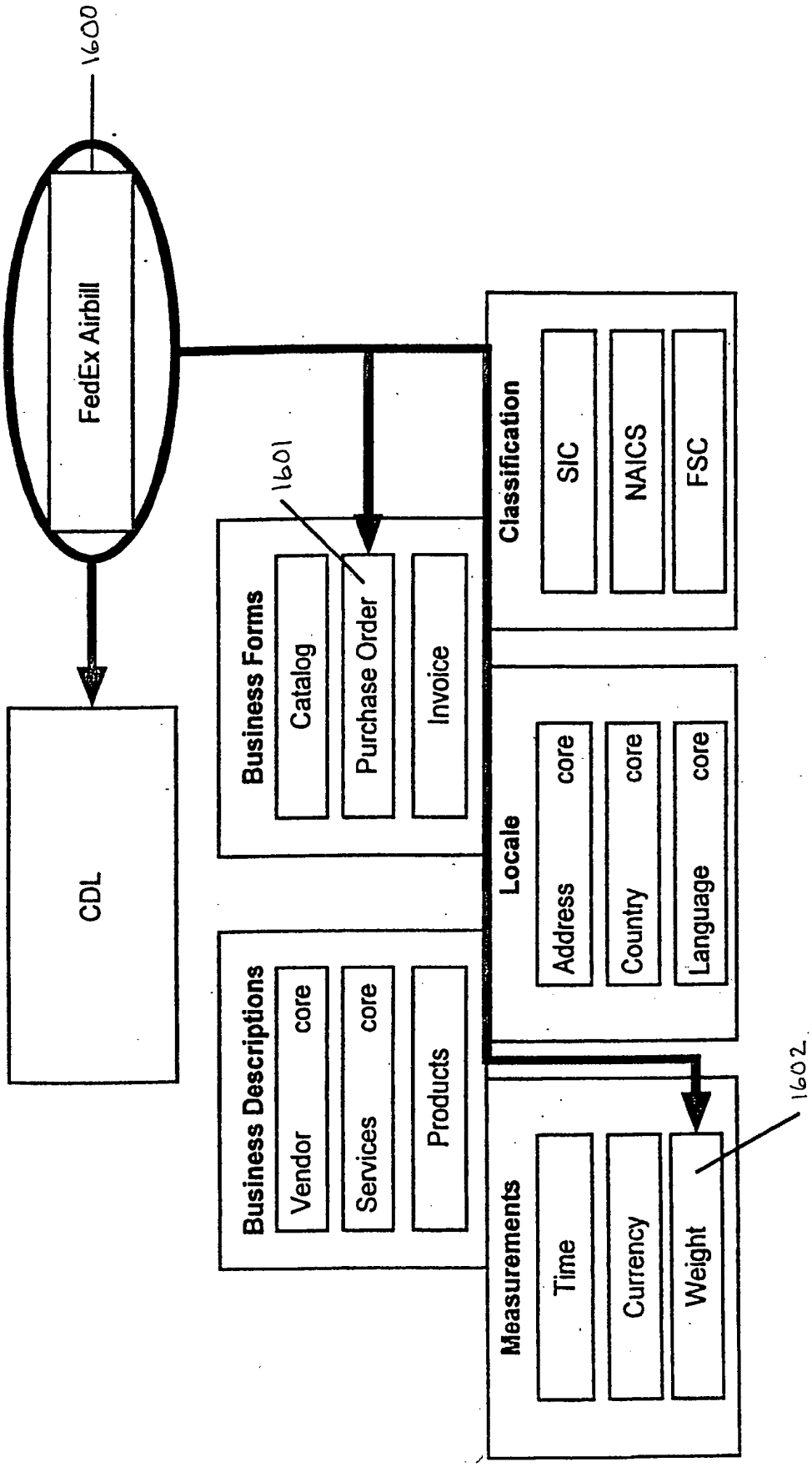

1600
FedEx Airbill
CDL
Business Forms
Catalog
Purchase Order
Invoice
1601
Business Descriptions
Vendor core
Services core
Products
Classification
SIC
NAICS
FSC
Locale
Address core
Country core
Language core
Measurements
Time
Currency
Weight
1602

FIG. 16

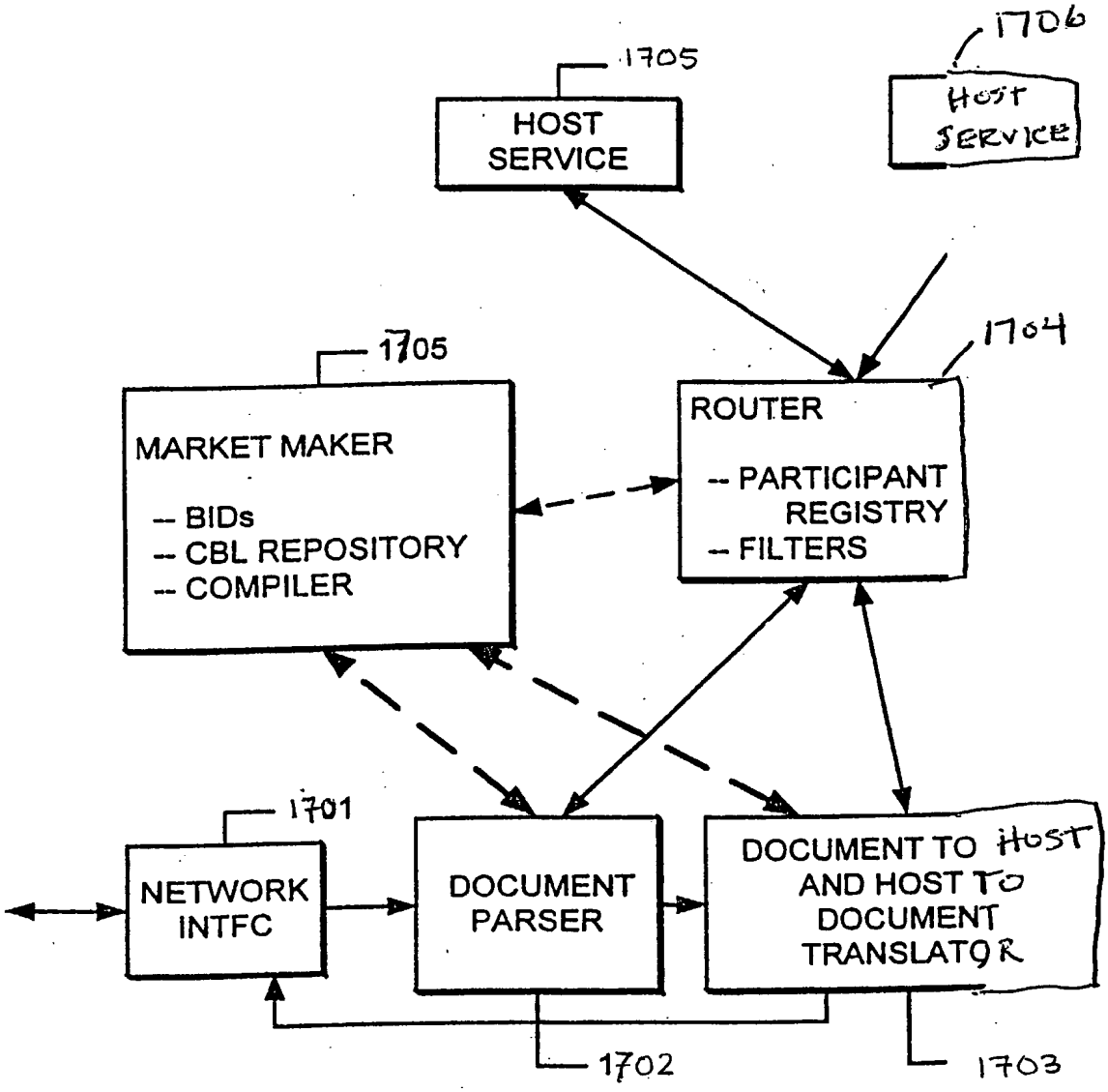

1706
HOST SERVICE
1705
HOST SERVICE
1705
MARKET MAKER
-- BIDs
-- CBL REPOSITORY
-- COMPILER
1704
ROUTER
-- PARTICIPANT REGISTRY
-- FILTERS
1701
NETWORK INTFC
DOCUMENT PARSER
DOCUMENT TO HOST AND HOST TO DOCUMENT TRANSLATOR
1702
1703

FIG. 17

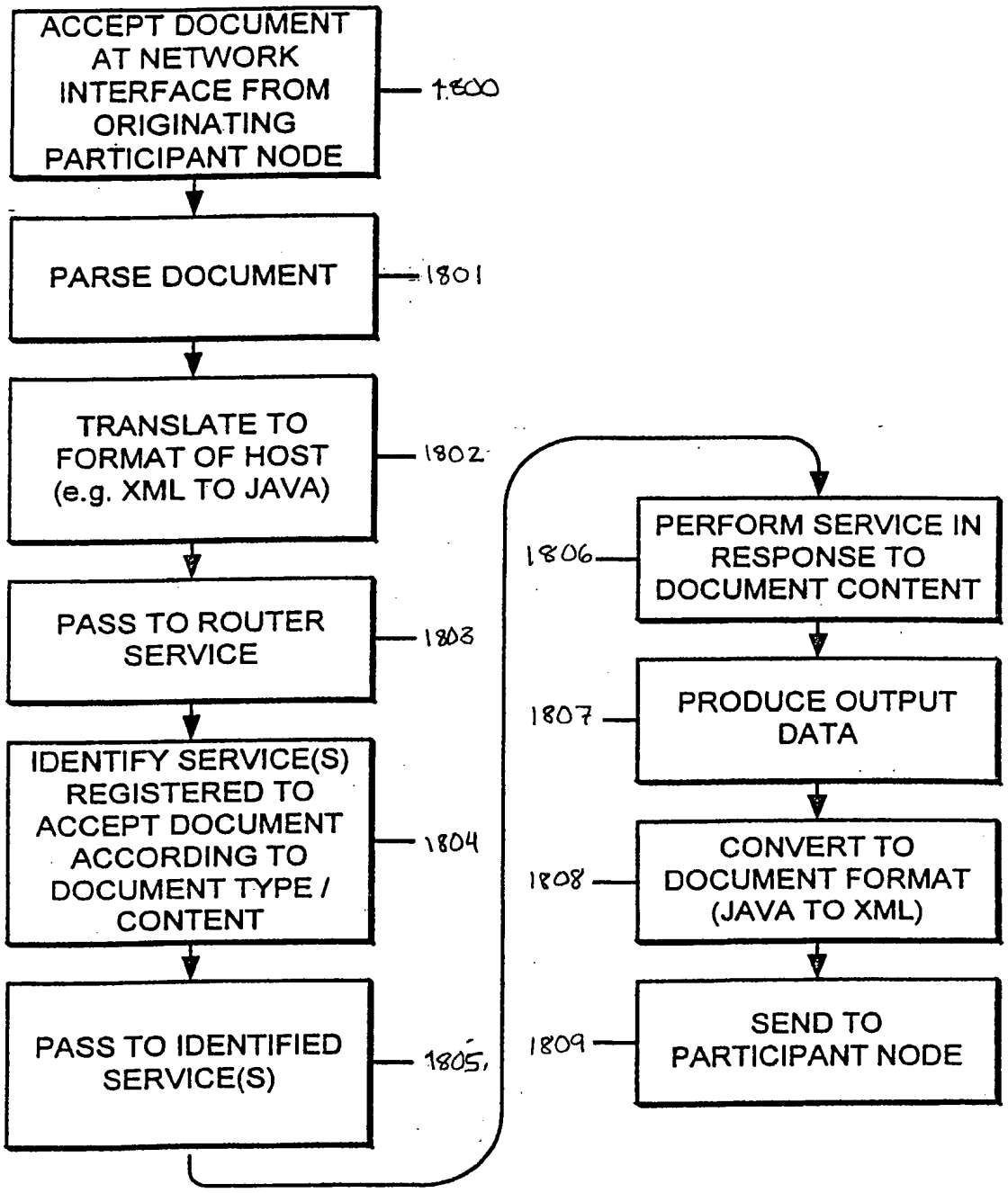

ACCEPT DOCUMENT AT NETWORK INTERFACE FROM ORIGINATING PARTICIPANT NODE
1800
PARSE DOCUMENT
1801
TRANSLATE TO FORMAT OF HOST (e.g. XML TO JAVA)
1802
PASS TO ROUTER SERVICE
1803
IDENTIFY SERVICE(S) REGISTERED TO ACCEPT DOCUMENT ACCORDING TO DOCUMENT TYPE / CONTENT
1804
PASS TO IDENTIFIED SERVICE(S)
1805
PERFORM SERVICE IN RESPONSE TO DOCUMENT CONTENT
1806
PRODUCE OUTPUT DATA
1807
CONVERT TO DOCUMENT FORMAT (JAVA TO XML)
1808
SEND TO PARTICIPANT NODE
1809

FIG. 18

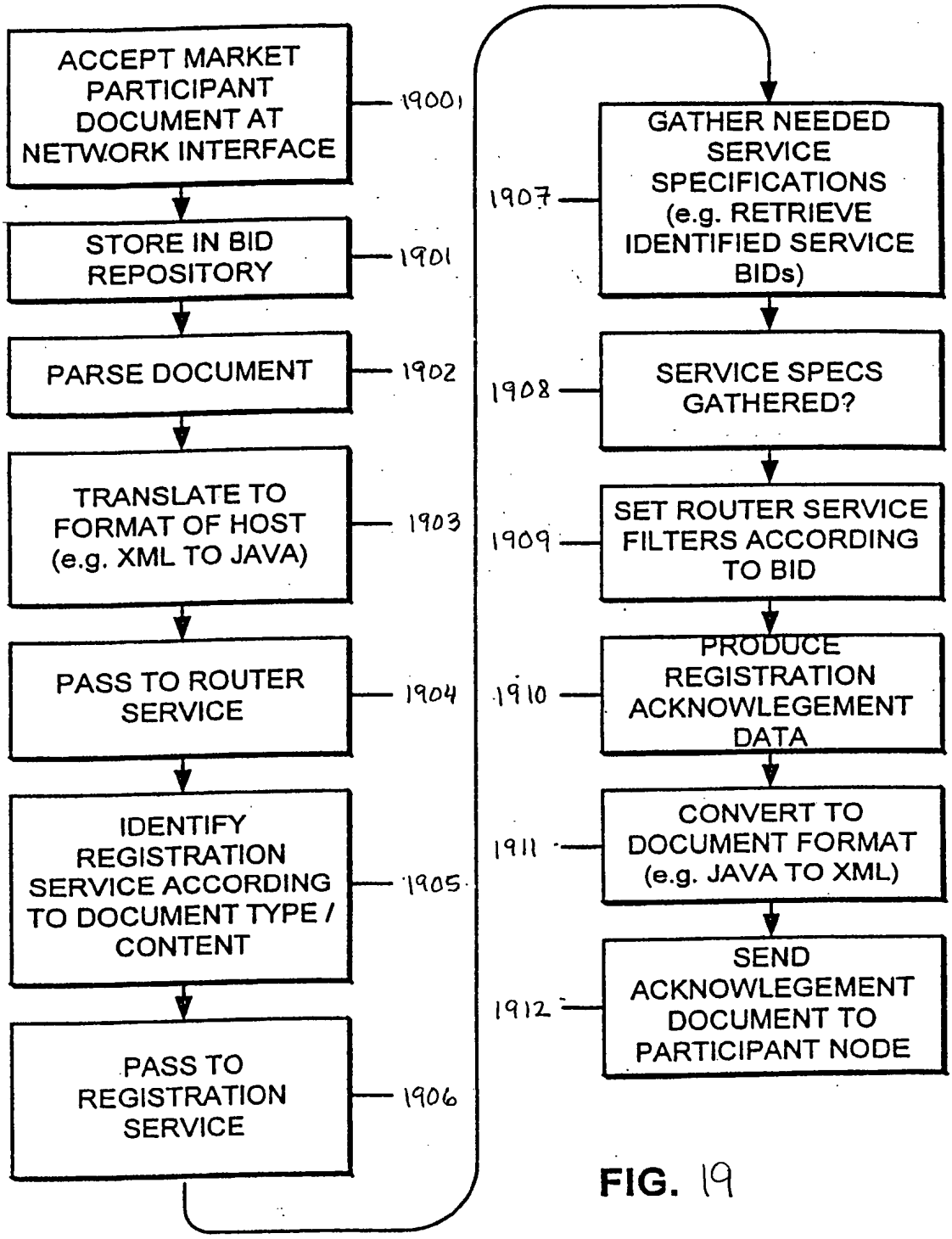

ACCEPT MARKET PARTICIPANT DOCUMENT AT NETWORK INTERFACE
1900
STORE IN BID REPOSITORY
1901
PARSE DOCUMENT
1902
TRANSLATE TO FORMAT OF HOST (e.g. XML TO JAVA)
1903
PASS TO ROUTER SERVICE
1904
IDENTIFY REGISTRATION SERVICE ACCORDING TO DOCUMENT TYPE / CONTENT
1905
PASS TO REGISTRATION SERVICE
1906
GATHER NEEDED SERVICE SPECIFICATIONS (e.g. RETRIEVE IDENTIFIED SERVICE BIDs)
1907
SERVICE SPECS GATHERED?
1908
SET ROUTER SERVICE FILTERS ACCORDING TO BID
1909
PRODUCE REGISTRATION ACKNOWLEGEMENT DATA
1910
CONVERT TO DOCUMENT FORMAT (e.g. JAVA TO XML)
1911
SEND ACKNOWLEGEMENT DOCUMENT TO PARTICIPANT NODE
1912

FIG. 19

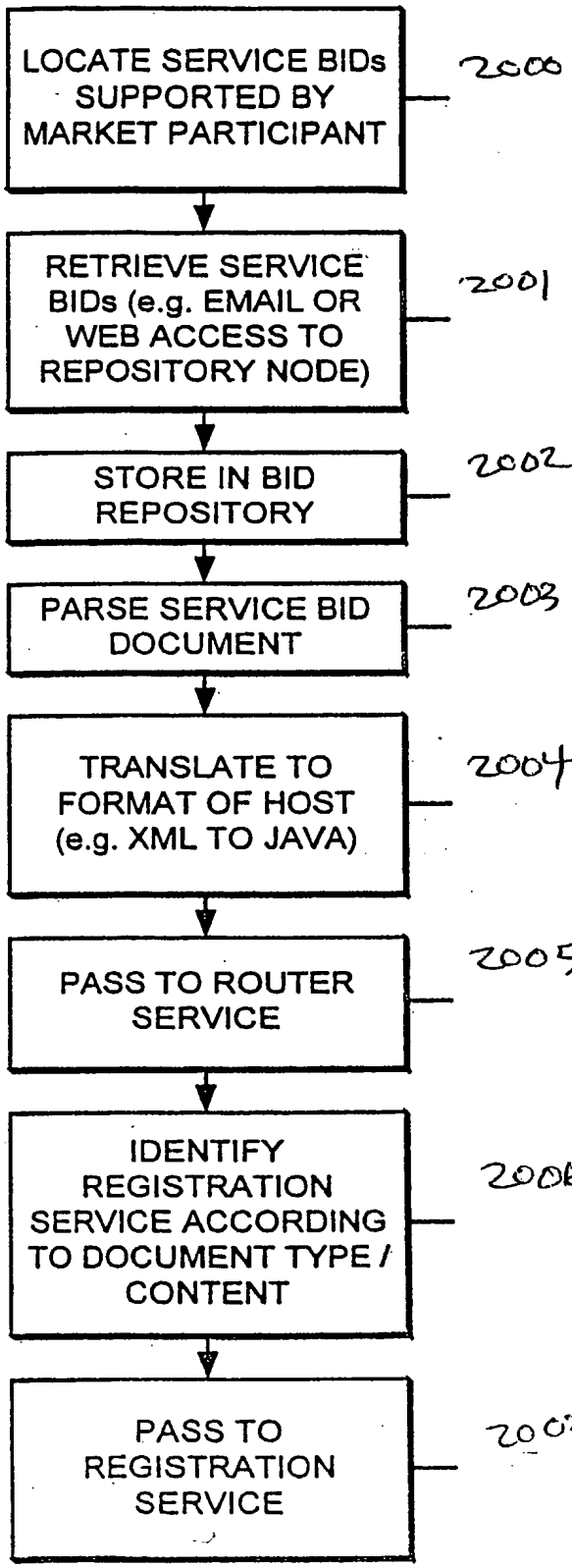

LOCATE SERVICE BIDs SUPPORTED BY MARKET PARTICIPANT
2000
RETRIEVE SERVICE BIDs (e.g. EMAIL OR WEB ACCESS TO REPOSITORY NODE)
2001
STORE IN BID REPOSITORY
2002
PARSE SERVICE BID DOCUMENT
2003
TRANSLATE TO FORMAT OF HOST (e.g. XML TO JAVA)
2004
PASS TO ROUTER SERVICE
2005
IDENTIFY REGISTRATION SERVICE ACCORDING TO DOCUMENT TYPE / CONTENT
2006
PASS TO REGISTRATION SERVICE
2007

Fig 20

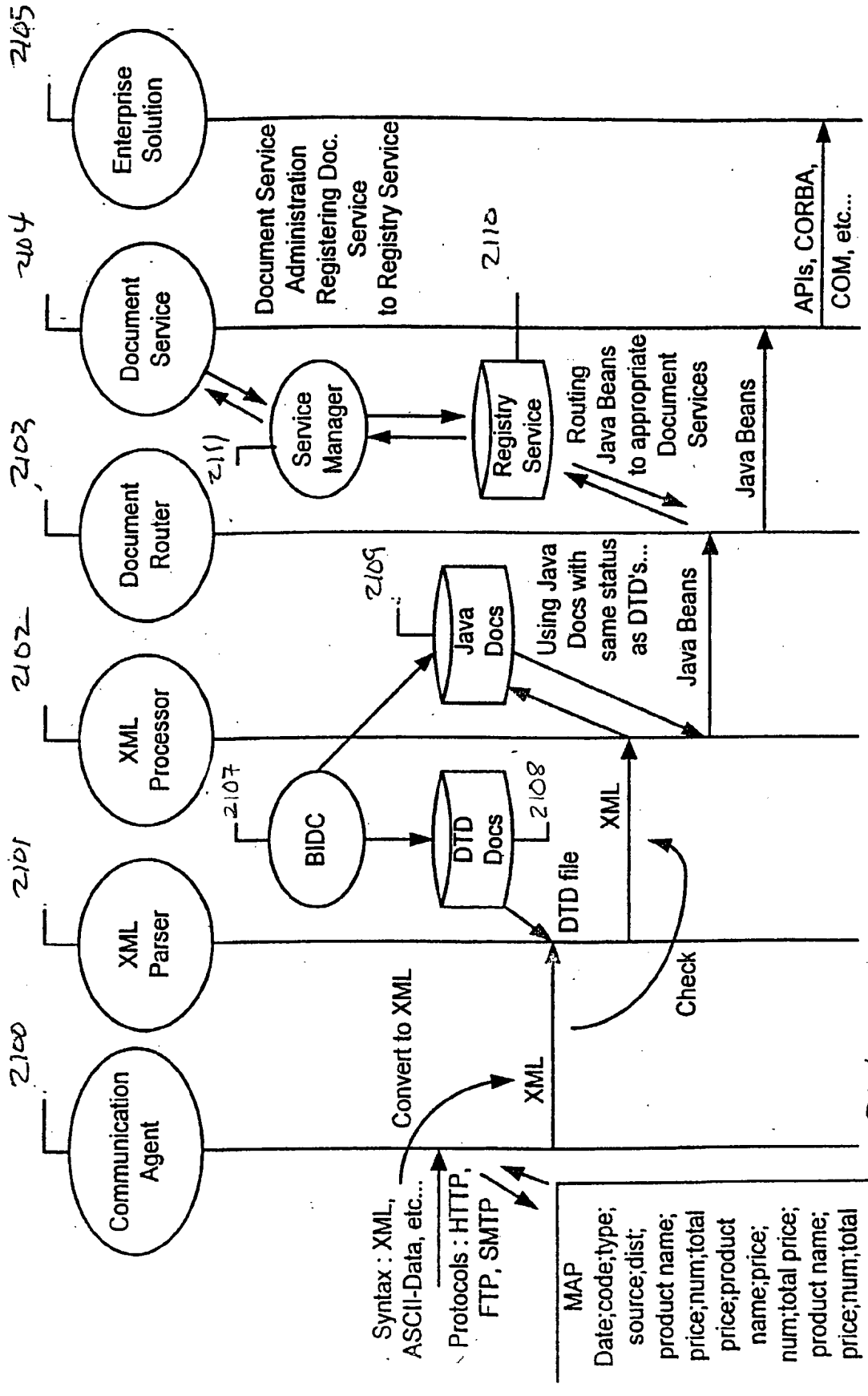

2100
2101
2102
2103
2104
2105
Communication Agent
XML Parser
XML Processor
Document Router
Document Service
Enterprise Solution
Syntax : XML, ASCII-Data, etc...
Protocols : HTTP, FTP, SMTP
Convert to XML
XML
Check
BIDC
2107
DTD Docs
2108
DTD file
XML
Java Docs
2109
Using Java Docs with same status as DTD's...
Java Beans
Service Manager
2111
Registry Service
2110
Routing Java Beans to appropriate Document Services
Java Beans
Document Service Administration Registering Doc. Service to Registry Service
APIs, CORBA, COM, etc...
2106
MAP
Date;code;type; source;dist; product name; price;num;total price;product name;price; num;total price; product name; price;num;total price;product name;price; num;total price;
Translation Table

FIG. 21

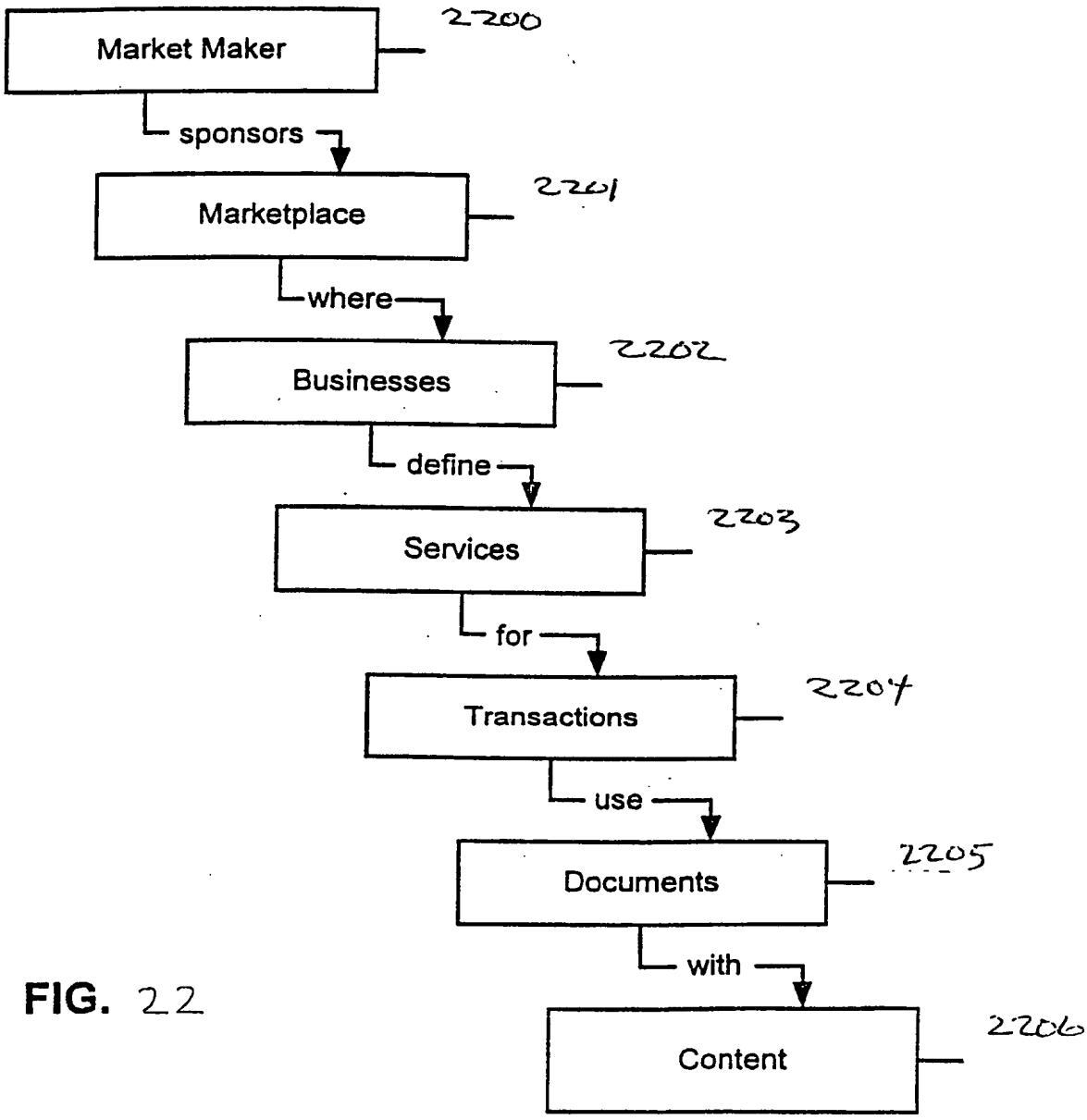

Market Maker
2200
sponsors
Marketplace
2201
where
Businesses
2202
define
Services
2203
for
Transactions
2204
use
Documents
2205
with
Content
2206

FIG. 22